



DEPARTAMENTO DE LENGUAJES Y SISTEMAS  
INFORMÁTICOS E INGENIERÍA DEL SOFTWARE

FACULTAD DE INFORMÁTICA

UNIVERSIDAD POLITÉCNICA DE MADRID

## DOCTORAL THESIS

# USABILITY-ORIENTED SOFTWARE DEVELOPMENT PROCESS

**AUTHOR**

LAURA CARVAJAL

**Ph.D. SUPERVISORS**

DR. ANA MARÍA MORENO

DR. MARÍA ISABEL SÁNCHEZ-SEGURA

2012



## **ABSTRACT**

Usability is the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions. Many studies demonstrate the benefits of usability, yet to this day software products continue to exhibit consistently low levels of this quality attribute. Furthermore, poor usability in software systems contributes largely to software failing in actual use.

One of the main disciplines involved in usability is that of Human-Computer Interaction (HCI). Over the past two decades the HCI community has proposed specific features that should be present in applications to improve their usability, yet incorporating them into software continues to be far from trivial for software developers. These difficulties are due to multiple factors, including the high level of abstraction at which these HCI recommendations are made and how far removed they are from actual software implementation. In order to bridge this gap, the Software Engineering community has long proposed software design solutions to help developers include usability features into software, however, the problem remains an open research question.

This doctoral thesis addresses the problem of helping software developers include specific usability features into their applications by providing them with a structured and tangible guidance in the form of a process, which we have termed the Usability-Oriented Software Development Process. This process is supported by a set of Software Usability Guidelines that help developers to incorporate a set of eleven usability features with high impact on software design.

After developing the Usability-oriented Software Development Process and the Software Usability Guidelines, they have been validated across multiple academic projects and proven to help software developers to include such usability features into their software applications. In doing so, their use significantly reduced development time and improved the quality of the resulting designs of these projects. Furthermore, in this work we propose a software tool to automate the application of the proposed process.

In sum, this work contributes to the integration of the Software Engineering and HCI disciplines providing a framework that helps software developers to create usable applications in an efficient way.



## TABLE OF CONTENTS

Chapter 1. Introduction	12
1.1 Importance of usability	12
1.2 Human-computer interaction recommendations	13
1.3 The Problem	18
1.4 Thesis structure	19
Chapter 2. Related Works	20
2.1 Introduction	20
2.2 Review Protocol	21
2.2.1 Research questions	21
2.2.2 Search process	21
2.2.2.1 Search terms and key phrases	21
2.2.3 Study selection	21
2.2.3.1 Inclusion criteria	21
2.2.3.2 Exclusion criteria	21
2.2.4 Data Extraction	22
2.2.5 Results	22
2.3 Relationship between usability and software architecture	26
2.3.1 Software Architecture Analysis of Usability	26
2.3.2 Analysing the impact of usability on software design	27
2.4 Architectural patterns	29
2.4.1 Linking usability to software architecture patterns through general scenarios	29
2.4.2 A Software Architectural View of Usability Patterns	30
2.4.3 Bringing usability concerns to the Design of Software architecture	32
2.4.4 Reconciling usability and interactive system architecture using patterns	34
2.4.5 A Responsibility-Based Pattern Language for Usability-Supporting Architectural Patterns	35
2.5 Systematic literature review results	36
Chapter 3. Hypothesis and approach to solution	38
3.1 Introduction	38
3.2 Hypothesis	38
3.3 Approximation to solution	39
3.3.1 Functional Usability Features	39
3.3.2 Usability-oriented software development process overview	40
3.3.3 Usability Guidelines for Software Development	41
Chapter 4. Usability-oriented software development process	43

4.1 Introduction	43
4.2 Process overview	43
4.3 Process detail	45
4.3.1 Requirements elicitation and analysis for usability	45
4.3.1.1 Functional usability requirements elicitation	46
4.3.1.2 Usability use case modeling	52
4.3.1.3 Identification of system responsibilities for usability	54
4.3.2 OO Software design activities for usability	55
4.3.2.1 Identification of high-level design component responsibilities for usability	56
4.3.2.2 Identification of low-level design component responsibilities for usability	58
4.3.2.3 Object Oriented software design for usability	60
4.4 Process Automation	62
4.4.1 Pre-load application	62
4.4.2 Support Tool for Usability-Oriented Development	65
Chapter 5. Usability Guidelines for Software Development	70
5.1 Introduction	70
5.2 “Undo” Usability Guideline for Software Development	71
5.2.1 Usability Guideline for Software Development: Analysis artifacts	71
5.2.1.1 Usability Elicitation Guideline	71
5.2.1.2 Usability Elicitation Clusters Clusters	75
5.2.1.3 Use Case Meta-model	77
5.2.1.4 System Responsibilities for Usability	78
5.2.2 Usability Guideline for Software Development: Design artifacts	80
5.2.2.1 High-level Design Component Responsibilities	80
5.2.2.2 Low-level Design Component Responsibilities for MVC	83
5.2.2.3 Usability Software Design Meta-models	86
5.3 “Abort” Usability Guideline for Software Development	90
5.3.1 Usability Guideline for Software Development: Analysis artifacts	90
5.3.1.1 Usability Elicitation Guideline	90
5.3.1.2 Usability Elicitation Clusters	93
5.3.1.3 Use Case Meta-model	94
5.3.1.4 System Responsibilities for Usability	95
5.3.2 Usability Guideline for Software Development: Design artifacts	96
5.3.2.1 High-level Design Component Responsibilities	97
5.3.2.2 Low-level Design Component Responsibilities for MVC	99
5.3.2.3 Usability Software Design Meta-models	101
5.4 Usability Guideline: ‘Step-by-step’	105
5.4.1 Usability Guideline for Software Development: Analysis artifacts	105
5.4.1.1 Usability Elicitation Guideline	105
5.4.1.2 Usability Elicitation Clusters	107
5.4.1.3 Use Case Meta-model	109
5.4.1.4 System Responsibilities for Usability	110
5.4.2 Usability Guideline for Software Development: Design artifacts	112
5.4.2.1 High-level Design Component Responsibilities	112
5.4.2.2 Low-level Design Component Responsibilities for MVC	114
5.4.2.3 Usability Software Design Meta-models	117
5.5 “Progress Feedback” Usability Guideline for Software Development	122
5.5.1 Usability Guideline for Software Development: Analysis artifacts	122
5.5.1.1 Usability Elicitation Guideline	122
5.5.1.2 Usability Elicitation Clusters	124
5.5.1.3 Use Case Meta-model	126
5.5.1.4 System Responsibilities	127
5.5.2 Usability Guideline for Software Development: Design artifacts	128
5.5.2.1 High-level Design Component Responsibilities	128
5.5.2.2 Low-level Design Component Responsibilities for MVC	129
5.5.2.3 Usability Software Design Meta-models	130
5.6 “System Status Feedback” Usability Guideline for Software Development	132

5.6.1 Usability Guideline for Software Development: Analysis artifacts	132
5.6.1.1 Usability Elicitation Guideline	132
5.6.1.2 Usability Elicitation Cluster Map	135
5.6.1.3 Use Case Meta-model	135
5.6.1.4 System Responsibilities for Usability	136
5.6.2 Usability Guideline for Software Development: Design artifacts	137
5.6.2.1 High-level Design Component Responsibilities	137
5.6.2.2 Low-level Design Component Responsibilities for MVC	138
5.6.2.3 Usability Software Design Meta-models	140
5.7 “Warning” Usability Guideline for Software Development	142
5.7.1 Usability Guideline for Software Development: Analysis artifacts	142
5.7.1.1 Usability Elicitation Guideline	142
5.7.1.2 Usability Elicitation Cluster Map	144
5.7.1.3 Use Case Meta-model	145
5.7.1.4 System Responsibilities	146
5.7.2 Usability Guideline for Software Development: Design artifacts	147
5.7.2.1 High-level Design Component Responsibilities	147
5.7.2.2 Low-level Design Component Responsibilities for MVC	148
5.7.2.3 Usability Software Design Meta-models	151
5.8 “Multi-level Help” Usability Guideline for Software Development	153
5.8.1 Usability Guideline for Software Development: Analysis artifacts	153
5.8.1.1 Usability Elicitation Guideline	153
5.8.1.2 Usability Elicitation Clusters	155
5.8.1.3 Use Case Meta-model	157
5.8.1.4 System Responsibilities	158
5.8.2 Usability Guideline for Software Development: Design artifacts	159
5.8.2.1 High-level Design Component Responsibilities	159
5.8.2.2 Low-level Design Component Responsibilities for MVC	160
5.8.2.3 Usability Software Design Meta-models	161
5.9 “Commands Aggregation” Usability Guideline for Software Development	163
5.9.1 Usability Guideline for Software Development: Analysis artifacts	163
5.9.1.1 Usability Elicitation Guideline	163
5.9.1.2 Usability Elicitation Clusters	165
5.9.1.3 Use Case Meta-model	167
5.9.1.4 System Responsibilities for Usability	168
5.9.2 Usability Guideline for Software Development: Design artifacts	169
5.9.2.1 High-level Design Component Responsibilities	169
5.9.2.2 Low-level Design Component Responsibilities for MVC	170
5.9.2.3 Usability Software Design Meta-models	172
5.10 “Preferences” Usability Guideline for Software Development	175
5.10.1 Usability Guideline for Software Development: Analysis artifacts	175
5.10.1.1 Usability Elicitation Guideline	175
5.10.1.2 Usability Elicitation Clusters	177
5.10.1.3 Use Case Meta-model	179
5.10.1.4 System Responsibilities for Usability	180
5.10.2 Usability Guideline for Software Development: Design artifacts	181
5.10.2.1 High-level Design Component Responsibilities	181
5.10.2.2 Low-level Design Component Responsibilities for MVC	183
5.10.2.3 Usability Software Design Meta-models	185
5.11 “Favorites” Usability Guideline for Software Development	188
5.11.1 Usability Guideline for Software Development: Analysis artifacts	188
5.11.1.1 Usability Elicitation Guideline	188
5.11.1.2 Usability Elicitation Clusters	190
5.11.1.3 Use Case Meta-model	192
5.11.1.4 System Responsibilities	193
5.11.2 Usability Guideline for Software Development: Design artifacts	194
5.11.2.1 High-level Design Component Responsibilities	194
5.11.2.2 Low-level Design Component Responsibilities for MVC	196
5.11.2.3 Usability Software Design Meta-models	197

5.12 “Personal Object Space” Usability Guideline for Software Development	200
5.12.1 Usability Guideline for Software Development: Analysis artifacts	200
5.12.1.1 Usability Elicitation Guideline	200
5.12.1.2 Usability Elicitation Clusters	202
5.12.1.3 Use Case Meta-model	204
5.12.1.4 System Responsibilities	205
5.12.2 Usability Guideline for Software Development: Design artifacts	206
5.12.2.1 High-level Design Component Responsibilities	206
5.12.2.2 Low-level Design Component Responsibilities for MVC	207
5.12.2.3 Usability Software Design Meta-models	209
Chapter 6. Validation	215
6.1 Introduction	215
6.2 Hypothesis	215
6.3 Methods	216
6.4 Variables	217
6.4.1 Development Time	217
6.4.2 Resulting design quality	217
6.4.3 Perceived complexity of usability mechanisms	218
6.5 Subjects	218
6.6 Data	219
6.6.1 Development time data	219
6.6.2 Design quality data	220
6.6.3 Perceived complexity data	220
6.7 Analysis	221
6.7.1 Development time data analysis	222
6.7.1.1 Global Analysis	222
6.7.1.2 Analysis per development phase	224
6.7.1.3 Analysis phase by feature	227
6.7.2 Design quality data analysis	230
6.7.2.1 Global Analysis	230
6.7.2.2 Analysis per development phase	232
6.7.3 Perceived complexity data analysis	233
6.7.3.1 Global Analysis	234
6.7.3.2 Analysis per development phase	235
6.8 Results and Findings	239
6.9 Threats to validity	242
Chapter 7. Conclusions and Future Work	243
7.1 Introduction	243
7.2 Conclusions	243
7.3 Future Work	245
Chapter 8. References	247
Chapter 9. Appendixes	249
9.1 Example application of the proposed Usability-Oriented Software Development Process	249
9.1.1 Requirements elicitation and analysis for usability	249
9.1.2 Functional usability requirements elicitation	249
9.1.3 Usability use case modeling	250
9.1.4 Identification of system responsibilities	251
9.2 OO software design activities for usability	251
9.2.1 Identification of High-level design component responsibilities for usability	251
9.2.2 Identification of Low-level design component responsibilities for usability	252
9.2.3 Object Oriented software design for usability	252
9.3 Full results of systematic literature review	255
9.4 Functional Usability Features. Color codes.	265



9.5 System Requirement Specifications	266
9.5.1 Online task manager (Gestor de Tareas Online)	266
9.5.1.1 Software Requirements	266
9.5.2 Home automation system (sistema de domótica del hogar)	268
9.5.2.1 Front-End	269
9.5.2.2 Back-end	271
9.5.3 Auction site	272
9.5.3.1 Alta Usuario Externo	272
9.5.3.2 Identificación	273
9.5.3.3 Edición de Perfil	274
9.5.3.4 Visualización de Perfil	274
9.5.3.5 Búsqueda de Productos	275
9.5.3.6 Puja en Subasta	275
9.5.3.7 2.3.2.7 Compra de Producto en Oferta	276

# CHAPTER 1. INTRODUCTION

## 1.1 Importance of usability

The term “usability” has been defined in a variety of ways throughout literature. In spite of the differences among these definitions, most of them categorize usability as a quality attribute that is desirable in a product and makes it easy to use.

One popular definition, offered by Jakob Nielsen, who is a leading usability researcher and consultant, states that:

“usability is a quality attribute that assesses how easy user interfaces are to use” [38]

He adds that “the word ‘usability’ also refers to methods for improving ease-of-use during the design process” and further breaks down the concept into the following five quality components:

- **Learnability:** How easy is it for users to accomplish basic tasks the first time they encounter the design?
- **Efficiency:** Once users have learned the design, how quickly can they perform tasks?
- **Memorability:** When users return to the design after a period of not using it, how easily can they reestablish proficiency?
- **Errors:** How many errors do users make, how severe are these errors, and how easily can they recover from the errors?
- **Satisfaction:** How pleasant is it to use the design?

Another widely-known definition of usability is the one offered by the ISO 9126 standard for product quality in software engineering since 1991, which also classifies usability as a quality attribute and defines it as follows:

“Usability is the capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions” [16]

This standard decomposes usability in understandability, learnability, operability, attractiveness and usability compliance, and is further expanded by its superseding version, the ISO 25010 issued in 2011, to include the following final set of sub-characteristics:

- **Appropriateness recognizability:** degree to which users can recognize whether a product or system is appropriate for their needs.
- **Learnability:** degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use.
- **Operability:** degree to which a product or system has attributes that make it easy to operate and control.
- **User error protection:** degree to which the system protects users against making errors
- **User interface aesthetics:** degree to which the user interface enables pleasing and satisfying interaction for the user.
- **Accessibility:** degree to which a product or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use.

Separately, the ISO 9241-11 standard for Ergonomics and Human-Computer interaction in 1998 states that:

“Usability is the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use” [28]

Yet another highly recognized standard, the IEEE Std. 1061 for Software Quality Metrics Methodology issued in 1998, offers the following definition:

“Usability is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component.” [26]

While the first two approaches focus mainly on the usability of the product, regarding whether it has the characteristics needed to make it easy to use, the last two have a wider scope, covering also the process and results of using the product.

Many other definitions and decompositions of usability can be found in literature. Kumar Dubey et al. compile over two dozen of them in [34] spanning multiple disciplines and three decades of research results, prompting works like that of Seffah et al. [40] who point out the difficulty of approaching usability in practice due to this wide variety of definitions.

Regardless of this seeming lack of consensus among the disciplines involved in nailing down the specifics of a definition for usability, it’s safe to say that they all agree in the fact that it is a critical aspect in interactive software systems [32]. In fact, its relevance is made evident by several studies [33][10][18][23][44], which demonstrate usability’s many benefits, including reduction of documentation and training costs, improvement of productivity, increase in morale and e-commerce revenue, and more. Accordingly, large-scale companies like IBM and Boeing Co. have begun to consider usability a key factor to consider when developing and buying software [31].

## 1.2 Human-computer interaction recommendations

One of the main disciplines involved in usability is that of Human-computer interaction (HCI). As defined by the Special Interest Group on Computer Human Interaction (SIGCHI), HCI is the “discipline concerned with the design, evaluation and implementation of interactive computing systems for human use and with the study of major phenomena surrounding them” [46].

SIGCHI further defines HCI as an interdisciplinary area, rooted in computer science (application design and engineering of human interfaces) working together with the following supporting disciplines: psychology (the application of theories of cognitive processes and the empirical analysis of user behavior), sociology and anthropology (interactions between technology, work, and organization), and industrial design (interactive products).

Over the past two decades, the HCI community has proposed specific recommendations for features that should be present in software systems in order to improve their usability [31]. In spite of overlaps and varied terminology, contributions such as Nielsen's design heuristics [38], Constantine and Lockwood's usability principles [16], Welie's patterns for interaction design [49] and Tidwell's usability patterns [42], to name a few, provide valuable guidelines on how to design usable GUI.

The following two examples are part of Welie's and Tidwell's collections respectively, and illustrate what these HCI recommendations look like and the different kinds of usability features they might cover.

The first example, shown in Figure 1.2-1, shows Jennifer Tidwell's Grid of Equals usability pattern. This pattern has five sections:

- **What:** States what the pattern does, in this case, arranging items in a grid on screen.
- **Use when:** Describes the context in which this pattern should be applied. In this example, the Grid of Equals pattern is applicable when content items have similar style and importance.
- **Why:** Explains the rationale behind the pattern. For this example, representing components in a grid announces that they have equal importance
- **How:** Describes the manner in which the interface items are to be arranged so that they conform to the pattern. In this example, decisions must be made regarding the size of the images, what they link to, the size of the text within them, etc.
- **Examples:** Provides examples of real-life applications that apply this pattern, as is the case of Nike.com and others in this example.

## Grid of Equals

### What

Arrange content items in a grid or matrix. Each item should follow a common template, and each item's visual weight should be similar. Link to jump pages as necessary.



### Use when

The page contains many content items that have similar style and importance, such as news articles, blog posts, products, or subject areas. You want to present the viewer with rich opportunities to preview and select these items.

Compare to: Center Stage

### Why

A grid that gives each item equal space announces that they have equal importance. The common template for items within the grid tells the user that the items are similar to each other. Together, these techniques establish a powerful visual hierarchy that should match the semantics of your content.

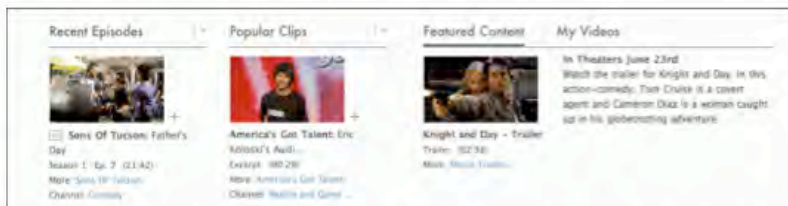
Grids look neat, ordered, and calming. That may suit the style of your site or app.

### How

Figure out how to lay out each item in the grid. Do they have thumbnail images or graphics? Headlines, subheads, summary text? Links to jump pages (e.g., a page with the full story)? Render them with more than just blocks of body text: make headlines of different colors, be creative with whitespace, and use images if you can do so evenly across all items. Experiment with ways to fit all the right information into a relatively small space— tall, wide, or square—and

## Examples

Hulu (below), CNN (below), and Nike (above) use a rigid template for each item. The overall effect is rhythmic and calming. Note how each site uses a different balance of text and imagery.



The examples from MapQuest and IBM show how to do this with only a single row of items. (Technically it's still a "grid.") The consistent visual treatment marks these items as peers of each other. Each item ends with one or more links—and that's true of the Hulu and CNN examples, too. Most of the examples I've seen of this pattern use it to showcase linked content.

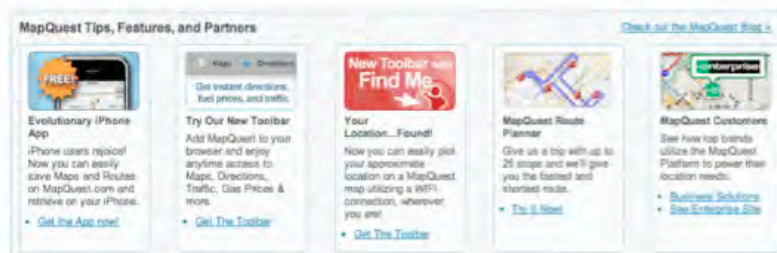


Figure 1.2-1 HCI Recommendation example. "Grid of Equals". Jennifer Tidwel [42]

The second example, shown in Figure 1.2-2, shows Welie's Processing page pattern. Welie's patterns have the same sections as Tidwel's, save for Tidwel's "What" section, which is represented as two separate fields in Wellie's patterns: Problem and Solution.

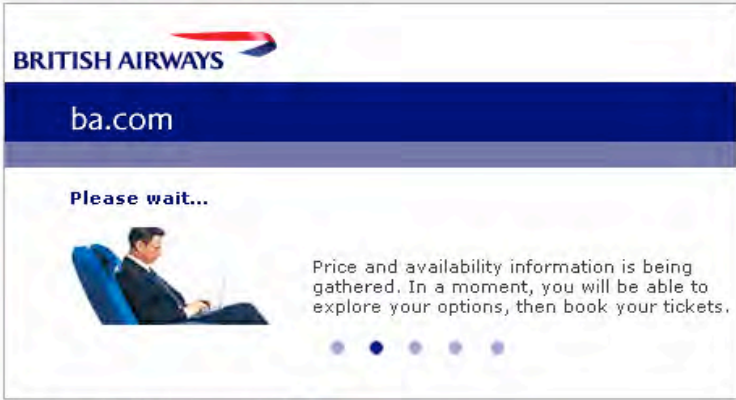
## Processing Page

### Problem

Users need feedback that their action is being performed but may take a while to complete

### Solution

Provide a feedback page with animation



The screenshot shows the British Airways website's processing page. At the top is the British Airways logo and the URL 'ba.com'. Below that, it says 'Please wait...' with an image of a man sitting in a blue airplane seat. To the right of the image, it says 'Price and availability information is being gathered. In a moment, you will be able to explore your options, then book your tickets.' Below this text is a progress indicator consisting of five dots, with the second dot from the left filled in blue.

From [www.british-airways.co.uk](http://www.british-airways.co.uk)

### Use when

You are designing a site where slow back-end systems are connected to. Some requests to the back-end system may take 5 to 30 seconds to complete and the users need some feedback telling them that their request is being performed and that they'll have to wait a bit. Only use this pattern when it is not possible to speed up the back-end processing time. Typically, a [Travel Site](#) using this when flight-availability is being looked-up. It also occurs frequently in a [Web-based Application](#)


### How

Provide information about the reason for the slow response so that users can have understanding of the problem. Also add an animation or real progress feedback so that users get a sense of continuity or progress.

### Why

Although it would be best to provide real progress feedback, this is often not technically possible in a web environment. Providing this type of feedback is the least that should be done for users who need to wait.

### More Examples



The screenshot shows the Expedia.com website's processing page. At the top is the Expedia logo and the URL 'Expedia.com'. Below that, it says 'We are processing your request...' in orange text. Below this text is a progress indicator consisting of five small squares, with the first four squares filled in orange and the fifth square filled in blue.

Figure 1.2-2 HCI Recommendation example. "Processing Page". Welie [49]

The above example details Welie's "Processing Page" pattern and it has the following parts:

- **Problem:** The problem that this pattern addresses, in this case the user's need for feedback while his request is being processed
- **Solution:** The author's proposed solution. In this example it's providing visual feedback information to the user
- **Use when:** Similarly to Tidwel's patterns, Welie's detail the situations when these patterns are useful. For this example it's when the application connects to an external system and communications may take more than a few seconds to be established
- **How:** Explains what should be present in the interface. In this case, an animated progress indicator, ideally showing the actual progress of the communications.
- **Why:** The reasons for providing this type of feedback
- **More examples:** As in Tidwel's case, the pattern ends with one or more real-life applications of this pattern.

As can be seen in these examples, the types of patterns offered by the HCI community are very comprehensive in terms of the usability problems that developers can encounter when developing applications and what these applications should look like in order to solve them. It can also be observed from these two examples alone, that the HCI patterns tackle a wide range of usability features that can be classified in different groups. For example, "Grid of Equals" deals solely with interface elements and how to lay them out visually to convey the right message, while "Processing page" deals more with the application functionality and implementing it may require the interface to communicate with underlying software components to determine the actual progress to be shown.

Juristo, Moreno and Sanchez examine a wide variety of such HCI recommendations for usability in [31]. In this work, the authors categorize these recommendations in three different groups, depending on their effect on software development.

The first proposed group is made up of usability recommendations whose impact is limited to the user interface, affecting only the system presentation through buttons, pull-down menus, check-boxes, background color, fonts, etc. Such a recommendation may suggest changing the color of certain interface elements or the size of a font to emphasize certain aspects of interest for the user. The authors suggest that such changes would be confined to altering the value of some source code variable related to the window or buttons, involving only slight modifications to the UI design, having no impact on the system core.

The second group is made up of usability recommendations that have an impact on the development process. These are recommendations that are not limited to specific software system products, but that require modifications to the development process in itself. Such is the case of "involving the user in software construction", as proposed by [24] and [25]. These kinds of recommendations would require making the development process more user-centered, including more powerful elicitation techniques, etc.

The third and final group proposed by the authors are recommendations that have an impact on software design. These recommendations involve building certain functionalities into the software to improve user-system interaction. For example, features like cancel an ongoing task [11], undoing a task [49][36][45], and receiving feedback on what is going on in the system [25][45] are examples of this type of HCI recommendations.

The authors have termed this last group of HCI recommendations Functional Usability Features, as they describe functionalities that the software should provide for the user. The focus of this doctoral thesis will be on this group, as the recommendations within it have the greatest impact on software design and, as shown in [31], incur the highest re-work costs when not considered in a timely manner during application development.

### 1.3 The Problem

In spite of the demonstrated rewards of developing usable software, to this day software products continue to exhibit consistently low levels of usability [41][10][32]. What is more, poor usability in software systems is likely the single largest reason that they fail in actual use [42].

Two possible groups of factors may explain the low usability of most current systems. The first group is made up of business-related factors that may range from market pressures to deliver software products before they are fully ready, to cost issues and even unawareness of the importance of usability in software development. The second group has to do with the technical difficulties that arise during software development when incorporating usability features into a system [5]. This last group of difficulties will be addressed within this work.

Even as the Software Engineering community has long struggled to consistently transform the usability principles proposed by the HCI community into actual software code over the past two decades, incorporating them into software is not a trivial task. As was shown in the second usability pattern example above in Figure 1.2-2, the HCI field may recommend that in order to solve the usability need of “providing the user with information on the action they are performing” the application may need to show “real progress feedback”. While this is a very sound recommendation, it is also very abstract from a software engineering point of view, in that it doesn’t really provide information on how this is to be done.

Another example of why incorporating usability into software is not a trivial task, especially in the case of functional usability features, is presented in [32]. The authors explain the task of building a cancel functionality for specific commands into an application. HCI experts [43][39] suggest that users should be able to interrupt time-consuming operations with no side-effects. In order to do so, the software application would need to be equipped with the functionality to gather information such as the data modifications that the action triggers, the state of the system prior to its execution, etc. It would also need to be able to restore the system to said previous state after cancellation, reallocate resources that were otherwise occupied while executing the action, etc. Such an example shows that this type of HCI recommendation has an effect that goes far beyond minor changes to the user interface and well into software design rework, making them complex to implement without further guidance than that stated in a few lines of text describing what it should look like on the GUI.

In their work, Seffah et al. [40] also refer to the complexities faced by software developers when attempting to include usability features with impact on design into their applications. The authors explain how usability features such as feedback information, application status and error messages are not trivial to implement, emphasizing the tight relationship between usability and the underlying software architecture.

As the results of the literary review of related works will show in Chapter 2, many attempts have been made to bridge the gap between these usability recommendations made by the HCI community and their consistent, reliable inclusion into software applications. While many valuable contributions have been made in this regard, the problem still remains an open research topic.

This work is focused on addressing this problem by providing software developers with structured, tangible guidance for including functional usability features into their software applications. The help provided to developers with this approximation to a solution spans multiple phases of the software development process, yet its main contribution is centered in the software design phase(s) of a project.



## 1.4 Thesis structure

In the following pages, **Chapter 2** describes the related works and existing approaches to solving the problem of helping software developers include usability into their applications. This chapter explains how the systematic literature review was performed and how the selection of these works was carried out.

**Chapter 3** presents the proposed hypothesis for this research and provides an overview of the proposed solution to the problem.

In the two chapters that follow, the proposed solution is described in detail. This solution is composed of two elements. The first is what we have termed the Usability-Oriented Software Development Process, described in **Chapter 4**. The second, presented in **Chapter 5**, are our proposed Usability Guidelines for Software Development.

After detailing the proposed solution, **Chapter 6** explains how this solution was put to the test in multiple experiments in order to validate it for conformance to the proposed hypotheses.

Once experimentation is carried out and its results are analyzed, **Chapter 7** presents our conclusions for the present work, as well as the opportunities it opens up for future lines of research.

Finally, **Chapter 8** lists all the works referenced within this doctoral thesis and **Chapter 9** presents all relevant appendixes.

This volume is accompanied by a CD containing additional material referenced in the Validation chapter.

## CHAPTER 2. RELATED WORKS

### 2.1 Introduction

The present work undertakes the open research question of providing software developers with guidelines to include functional usability features into their software. However, before attempting to propose a solution a thorough review of existing literature relevant to this research topic was conducted. The purpose of this review was to:

- Gain a wider perspective of the subject matter (software usability)
- Identify where and how the problem to address lies within this subject matter and how it might relate to other proposed solutions, if any exist.
- Clearly define the scope of the problem as it is to be addressed by this work, expanding or modifying the original research question
- Avoid reproducing documented unviable solutions or partial results, if any exist.
- Keep up to date with possible new contributions related to the problem being addressed.

In order to conduct this review rigorously and reliably, this work follows Kitchenham's methodology for Systematic Literature Reviews for Software Engineering [34].

A Systematic Literature Review, as defined by Kitchenham in [34] is "a means of identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area, or phenomenon of interest."

Though originally intended for reviewing research results in the medical field [15], in her work, Kitchenham modifies the existing methodology and adapts it to the Software Engineering field. By following her guidelines, researchers in the Software Engineering field can compile existing information about a particular research topic in a repeatable, reliable and unbiased manner.

Section 2.2 below presents the protocol that was observed during the systematic literature review for this work, as well as the resulting list of chosen research publications. Later, in sections 2.3 and 2.4, these works are presented in detail. Finally, section 2.5 presents a summarized view of the results of the literature review and how they relate to the present work.

## 2.2 Review Protocol

The protocol for a systematic literature review specifies the research questions being addressed and the methods that will be used to perform the review.

### 2.2.1 Research questions

The purpose of this literature review was to find all publications that studied the relationships between usability software architecture and design.

### 2.2.2 Search process

The main search engine used for this systematic literature review is Google Scholar (scholar.google.com) which has access to a vast array of published books, as well as to the contents of the following digital libraries, among others:

- IEEExplore
- ACM Digital library
- Citeseer library
- Inspec
- ScienceDirect

These sources were of particular interest to the present review because they are known to collectively cover the most relevant sources for research results in the field.

#### 2.2.2.1 Search terms and key phrases

The following list of key phrases was used to perform the searches. It was designed to cover all aspects of the aforementioned research question

Table 2.2-1 Search key phrases

Software architecture usability
Usability patterns software design
Architectural patterns usability
Usability patterns architecture
Software engineering usability

It is worth noting that the above search terms were input into Google Scholar using the ‘with all of these words’ option. The ‘with any of these words’ option was discarded as repeated use did not yield any relevant results not already included in the first type of search.

### 2.2.3 Study selection

The following criteria were used to produce an initial list of selected publications. As the systematic literature review is an iterative process, these criteria were applied repeatedly throughout the development of this work to keep the resulting list up to date.

#### 2.2.3.1 Inclusion criteria

From the lists of studies that matched the key phrases above, those meeting the following criteria were automatically included for initial reviewing:

1. Studies focusing on providing any kind of design or architectural solutions to the problem of including aspects of usability into software
2. Any works that studied the relationship between software usability and software architecture and/or design

#### 2.2.3.2 Exclusion criteria

The following exclusion criteria helped trim down the initial list of studies to review:

1. Multiple publications of the same data (only the most complete and/or recent study was preserved)
2. Unpublished data
3. Papers written in languages other than English and Spanish
4. Lack of validation

#### 2.2.4 Data Extraction

Once the initial set of studies was established, the process of data extraction was conducted manually on physical media. With the exception of books, all studies were printed out, organized in a binder and relevant content either was highlighted (relevant text, author names, etc.) or written in (topic, strength of study, etc.). This process was conducted iteratively, adding new papers that matched, or didn't, respectively, the criteria above.

For each study, the following fields were considered:

- Title
- Source
- Publication date
- Author(s)
- Publication type (conference proceeding, paper, etc.)
- Topic area (architectural solutions, study of the architecture/usability relationship, etc.)

#### 2.2.5 Results

Five searches were performed in Google Scholar, one for each key phrase in the order shown in Table 2.2-1. The process followed determined that if a study was found using more than one set of keywords it was assigned to the first set with which it was found (duplicate results using subsequent sets of keywords were immediately excluded). In other words, from the second set of keywords onwards, only non-duplicate studies were gathered, explaining why the vast majority of studies are assigned to the first set of keywords: "software usability architecture".

Appendix 9.3 includes the full Google Scholar search results obtained for the key phrases that produced relevant results. It is worth noting that the fourth and fifth key phrases in Table 2.2-1 yielded no results that were a) not included in the results obtained from the first three searches, nor b) consistent with any of the inclusion criteria

After a first sweep of the raw results shown in Appendix 9.3 only those that appeared to meet the inclusion criteria (and to not meet the exclusion criteria) based on the title and abstract alone were preserved. These studies are shown in Table 2.2-2. After fully reviewing each of these studies, those that were found to meet any of the exclusion criteria and were further discarded. The nine studies that were ultimately kept are highlighted.

The reasoning behind the exclusion of each discarded study (the exclusion criterion by which they were left out or the inclusion criteria they initially appeared to meet but ultimately did not) is presented in Table 2.2-2.

Table 2.2-2 Selected studies

GOOGLE SCHOLAR RESULTS FOR key phrase #1: "software architecture usability"					
#	Authors	Title	Source	Year	Type <sup>1</sup>
1	Bass, L.; John, B. E.	Linking usability to software architecture patterns through general scenarios	Journal of Systems and Software	2003	JA
2	Folmer, E.; van Gurp J.; Bosch J.	Scenario-based assessment of software architecture usability	Bridging the Gaps Between Software Engineering and Human-Computer Interaction	2003	JA
3	Bass, L.; John, B. E.	Achieving Usability through software architecture	CMU/SEI Technical Report	2001	TR
4	Bass, L.; John, B. E.	Supporting usability through software architecture	IEEE Computer	2002	JA
5	Folmer, E.; van Gurp J.; Bosch J.	Software architecture analysis of usability	Engineering Human Computer Interaction and Interactive Systems	2005	JA
6 <sup>2</sup>	Golden, E.; Bass, L.; John, B. E.	The Value of Usability-Supporting Architectural Pattern in Software Architecture Design: A Controlled Experiment	International Conference on Software Engineering	2005	CP
7	Bass, L.; John, B. E.; Juristo, N.; Sanchez-Segura, M.I.; Adams, R.J.	Bringing Usability Concerns to the Design of Software Architecture	Lecture Notes in Computer Science	2005	JA
8	Bass, L.; John, B. E.; Kates, J.	Achieving Usability through software architectural styles	CHI 2000	2000	CP
9	Folmer, E.; Bosch J.	Usability patterns in software architecture	Human-computer interaction: theory and practice	2003	JA
10	Bosch, J.; Juristo, N.	Designing software architectures for usability	Proceedings of the 25th International Conference on Software Engineering. IEEE Computer Society	2003	TO
11	Folmer, E.; Bosch J.	Architecting for usability: a survey	Journal of Systems and Software	2004	JA
12	Juristo, N.; Lopez M.; Moreno A. M.; Sanchez M. I.	Improving software usability through architectural patterns	Bridging the Gaps Between Software Engineering and Human-Computer Interaction	2003	JA
13	Seffah, A.; Metzker E.	The obstacles and myths of usability and software engineering	Communications of the ACM	2004	JA
14	Adams, R.J.; Bass, L.; John, B. E.	Applying general usability scenarios to the design of the software architecture of a collaborative workspace	Human-Centered Software Engineering: Integrating Usability in the Software Development Lifecycle. Netherlands	2005	CP
15	Juristo, N.; Moreno, A.M.; Sanchez-Segura, M.I.	Analysing the impact of usability on software design	Journal of Systems and Software	2007	JA
16	Golden, E.; John B. E.; Bass L.	Quality vs. quantity: Comparing evaluation methods in a usability-focused software architecture modification task	International Symposium on Empirical Software Engineering	2005	CP
17	Juristo, N.; Moreno A. M.; Sanchez M. I.	Guidelines for Eliciting Usability Functionalities	IEEE TRANSACTIONS ON SOFTWARE ENGINEERING	2007	JA
18	Juristo, N.; Moreno A. M.; Sanchez M. I.	Clarifying the Relationship between Software Architecture and Usability	Proceedings of the Sixteenth International Conference on Software Engineering and Knowledge Engineering	2004	CP
19	Seffah, A.; Mohamed, T.; Habieb-Mammar, H.; Abran, A.	Reconciling usability and interactive system architecture using patterns	The Journal of Systems and Software	2008	JA
20	Golden, E.; Bass, L.; John, B. E.;	Helping software architects design for usability	ACM SIGSOFT	2009	CP

<sup>1</sup> JA = Journal Article, TR = Technical Report, CP = Conference Proceedings, TN = Technical Note, Tutorial

<sup>2</sup> Validation for results presented in study #7 addressed in section 2.4.3

GOOGLE SCHOLAR RESULTS FOR KEY PHRASE #2: "usability patterns software design"					
#	Authors	Title	Source	Year	Type
21	John, B. E.; Bass L.; Sanchez-Segura M. I.; Adams R. J.	Bringing usability concerns to the design of software architecture	Engineering Human Computer Interaction and Interactive Systems	2005	JA
22	Folmer, E.; van Gurp J.; Bosch J.	Scenario-based assessment of software architecture usability	Bridging the Gaps Between Software Engineering and Human-Computer Interaction	2003	JA
23	Ferre, X.; Jusisto N.; Moreno A. M.; Sanchez M. I.	A software architectural view of usability patterns	Proceedings of INTERACT	2003	JA
24 <sup>3</sup>	Golden, E.; John B. E.; Bass L.	The value of a usability-supporting architectural pattern in software architecture design: a controlled experiment	Proceedings of the 27th international conference on Software engineering ACM	2005	CP
25	Folmer, E.; van Gurp J.; Bosch J.	A framework for capturing the relationship between usability and software architecture	Software Process: Improvement and Practice	2003	JA
26	Nielsen, J.	The usability engineering life cycle	Computer IEEE	2002	JA
27	Bass, L.; John, B. E. Juristo, N.; Sanchez-Segura, M.I.	Usability-supporting Architectural Patterns	26 <sup>th</sup> International Conference on Software Engineering	2004	CP
GOOGLE SCHOLAR RESULTS FOR KEY PHRASE #3: "architectural patterns usability"					
#	Authors	Title	Source	Year	Type
28	Bass, L.; John, B. E.; Golden, E.; Stoll, P.	A responsibility-based pattern language for usability-supporting architectural patterns	Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems	2009	JA
29	Stoll, P.; John, B.E.; Bass, L.; Golden, E.	Preparing Usability Supporting Architectural Patterns for Industrial Use	Proceedings of the International Workshop on: Interplay between Usability Evaluation and Software Development	2008	CP

---

<sup>3</sup> Validation for results presented in study #7 addressed in section 2.4.3

Table 2.2-3 Reasoning for exclusion of studies in final list

#	Excluded Paper	Crit.	Reasoning
2	Scenario-based assessment of software architecture usability	ec <sup>4</sup> #1	More complete results are presented two years later in study #5 "Software architecture analysis of usability"
3	Achieving Usability through software architecture	ec #1	More complete results are presented four years later in study #7 "Bringing Usability Concerns to the Design of Software Architecture"
4	Supporting usability through software architecture	ec #1	More complete results are presented three years later in "Bringing Usability Concerns to the Design of Software Architecture"
8	Achieving Usability through software architectural styles	ec #1	More complete results are presented three years later in "Bringing Usability Concerns to the Design of Software Architecture"
9	Usability patterns in software architecture	ec #1	More complete results are presented two years later in study #5 "Software architecture analysis of usability"
10	Designing software architectures for usability	ic #1 ic #2	Though the abstract seemed to meet the inclusion criteria, the content was a short tutorial outline that in itself did not
11	Architecting for usability: a survey	ec #1	More complete results are presented one year later in study #5 "Software architecture analysis of usability"
12	Improving software usability through architectural patterns	ec #1	More complete results are presented one year later in study #23 "A software architectural view of usability patterns"
13	The obstacles and myths of usability and software engineering	ic #1 ic #2	Though the abstract seemed to meet the inclusion criteria, this paper is a more general analysis of software usability
14	Applying general usability scenarios to the design of the software architecture of a collaborative workspace	ic #1 ec #1	Only part of the results in this study are relevant to our topic, specifically those presented in a study of the same year: #7 "Bringing Usability Concerns to the Design of Software Architecture"
16	Quality vs. quantity: Comparing evaluation methods in a usability-focused software architecture modification task	ec #1	More complete results are presented four years later in study #7 "Bringing Usability Concerns to the Design of Software Architecture"
17	Guidelines for Eliciting Usability Functionalities	ic #1 ic #2	Though a precursor of this present work, this paper deals solely with the relationship between usability and requirements elicitation, with only brief allusions to design
18	Clarifying the Relationship between Software Architecture and Usability	ec #1 ec #4	Part of the results presented in this work are present in study #23 "A software architectural view of usability patterns" of the previous year. Additional results are in the very early stages and thus not validated.
20	Helping software architects design for usability	ec #1	More complete results are presented in study #29 "Preparing Usability Supporting Architectural Patterns for Industrial Use" of the same year
21	Bringing usability concerns to the design of software architecture	ec #1	More complete results are presented in study #7 "Bringing Usability Concerns to the Design of Software Architecture" of the same year
22	Scenario-based assessment of software architecture usability	ec #1	More complete results are presented two years later in study #5 "Software architecture analysis of usability"
25	A framework for capturing the relationship between usability and software architecture	ec #1	More complete results are presented two years later in study #5 "Software architecture analysis of usability"
26	The usability engineering life cycle	ic #1 ic #2	This study does not address software architecture directly, but rather a more general view of usability and diverse aspects of software development
27	Usability-supporting Architectural Patterns	ec #1	More complete results are presented one year later in study #7 "Bringing Usability Concerns to the Design of Software Architecture"
29	Preparing Usability Supporting Architectural Patterns for Industrial Use	ec #1	More complete results are presented one year later in study #28 "A responsibility-based pattern language for usability-supporting architectural patterns"

The selected studies were organized into two groups, depending on the subject matter they covered within our research question: Firstly, we have the studies that focus on analyzing the extent of the relationship between usability and software architecture, namely papers 5 and 15 above. Secondly, there are the studies that propose different kinds of architectural patterns to including usability into software 1, 7, 19, 23 and 28. Sections 2.3 and 2.4 describe each of the studies in these two groups, respectively.

<sup>4</sup> ec = "exclusion criteria that was met by the study", ic = "inclusion criteria that was not met by the study"

## 2.3 Relationship between usability and software architecture

The following sections detail the studies that have delved into determining the nature of the relationships that may exist between the usability needs of a software application and its architecture.

Arranged chronologically, the results presented herein provide substantial evidence of the quantifiable relationship between usability and software architecture. Furthermore, they highlight the importance of addressing usability issues as early as the design phase of the development process.

### 2.3.1 Software Architecture Analysis of Usability

*Folmer, E.; van Gurp, J.; Bosch, J. Journal Article. LNCS 2005 [20]*

In this work, the authors present an assessment technique to assist software architects in designing the architecture of their systems in a way that supports usability. This assessment technique (SLUTA, for Scenario based architecture Level Usability Analysis) promotes explicit evaluation of usability during architectural design, with the purpose of discovering usability issues during this early stage of development, as opposed to doing so during system maintenance to a higher cost. The authors point out that while this type of software architecture analysis contributes to the support of usability within the software architecture, it also highlights the limitations that a software architecture may impose on its intended level of usability.

The foundation for this assessment technique is a framework proposed by the authors that expresses the relationships between usability and software architecture. More specifically, this framework is made up of three layers:

- **Usability Attributes:** Complied from usability literature, these attributes represent the most common notions of usability, such as *learnability*, *efficiency*, *reliability*, etc.
- **Usability Properties:** These are heuristics and design principles that usability researchers have found to directly influence system usability. They represent the link between the usability attributes (above) and the usability patterns (below) proposed by the authors of this work. Such properties are high-level design primitives with known architectural implications, such as *providing feedback*, *consistency*, etc.
- **Architecture Sensitive Usability Patterns:** These patterns express potential architectural implications (rather than specific design solutions) that software developers will encounter in attempting to solve the problem posed by a specific pattern, such as *“user profiles”* or *“actions on multiple objects”*. They were identified by the authors from various case studies, existing applications and pattern collections and literature surveys.

The SALUTA technique itself consists of four steps, namely:

1. **Creating the usage profile.** This step entails identifying the users of the system, the tasks they will be expected to perform and the context of use. Furthermore, for each valid combination of the three, the authors quantify each usability attribute from their proposed framework described above, assigning them a priority for every scenario. Ultimately, this step in the technique results in a set of usage scenarios that express the required usability of the system to develop.
2. **Describing the provided usability:** This is where the information about the software architecture is collected, specifically, its support for usability, based to the proposed framework.
3. **Evaluating the scenarios:** Which is an evaluation of the support provided by the architecture for each of the scenarios in the usage profile.



4. **Interpreting the results:** The authors propose that the results of this technique can be useful during iterative developments. For example, in cases when the architecture has been identified as having poorly developed a certain needed usability property, said property can be improved in a future iteration. These results are also proposed as useful in comparing two architectures in regards of their usability, by looking at the ‘scores’ obtained by each for the scenarios being compared.

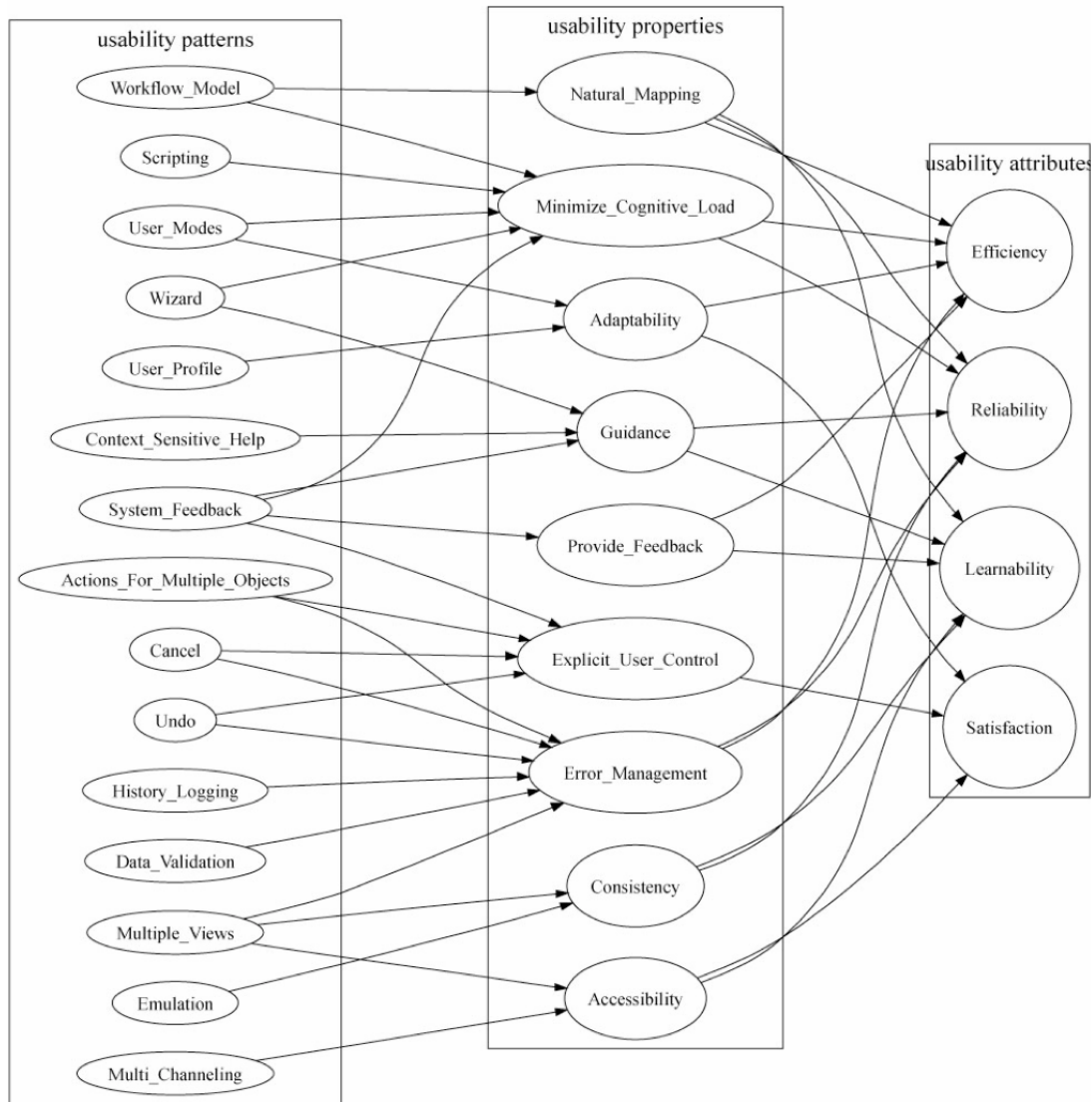


Figure 2.3-1 Partial view of the Usability Framework proposed by Folmer et al. 2005

SLUTA was evaluated over three industry projects, yielding encouraging results, though the need for further validation and additional case studies is suggested by the authors. The framework it is based upon, however, would benefit from a more case studies to determine its validity.

### 2.3.2 Analysing the impact of usability on software design

*Natalia Juristo, Ana Maria Moreno, Maria-Isabel Sanchez-Segura. JSS Vol 80/2007 [31]*

In this study, the authors look to determine, measure and quantify the effects of incorporating certain usability features into the design of a software application.

The authors begin by analyzing the Human-Computer Interaction (HCI) literature in order to pinpoint those usability heuristics or recommendations that might have an impact beyond the

user interface and into the software design. Then, by studying different real-world applications where these heuristics are included, the authors can determine what this impact entails in terms of new classes, methods and relationships in the application's design. With this data, the authors not only demonstrate the existence of a relationship between these usability recommendations and the software architecture, but are also able to estimate the implications of including these recommendations into a system.

After performing the HCI literature research and merging different authors' terminologies and recommendations, the authors produced a preliminary list of the usability features that they identified as having an impact on software design. This list along with the HCI author(s) who've addressed each usability feature in literature, is shown in Figure 2.3-2

Functional usability features
Feedback (Tidwell, 1999; Brighton, 1998; Coram and Lee, 1996; Welie, 2003; Tidwell, 2005; Nielsen, 1993; Constantine and Lockwood, 1999; Shneiderman, 1998; Hix and Hartson, 1993; Heckel, 1991)
Undo (Tidwell, 2005; Welie, 2003; Brighton, 1998)
Cancel (Tidwell, 2005; Brighton, 1998; Nielsen, 1993)
Form/field validation (Tidwell, 2005; Brighton, 1998; Shneiderman, 1998; Hix and Hartson, 1993; Rubinstein and Hersh, 1984; Constantine and Lockwood, 1999)
Wizard (Welie, 2003; Tidwell, 2005; Constantine and Lockwood, 1999)
User expertise (Tidwell, 1999; Welie, 2003; Hix and Hartson, 1993; Heckel, 1991)
Multi level help (Tidwell, 2005; Welie, 2003; Nielsen, 1993)
Use of different languages (Nielsen, 1993)
Alert (Brighton, 1998; Welie, 2003)

Figure 2.3-2 Preliminary list of usability features with impact on software design proposed by Juristo et al. 2007

In this work, the above usability features are termed Functional Usability Features (FUFs)

In order to determine the impact, if any, that these FUFs would have on software design, the authors had them applied over several university projects. For each project, the developers were asked to design their software without including any FUFs. After design was completed, they were asked to modify them to include each of the FUFs mentioned above.

One example is given, of a project that altered its design to include a particular flavor of the Feedback FUF, namely the System Status Feedback. After being given the list of responsibilities that the system must carry out in order to properly include this type of feedback, the developing team in question reported needing to include three new classes, five new methods and four new associations to their existing designs. Six other groups reported similar results, from which the authors were able to quantify the following criteria, based on standard metrics used to measure object-oriented systems complexity:

- FUF Impact on system functionality
- Number of new classes derived from including the FUF
- Complexity of new methods derived from including the FUF
- Number of new interactions stemming from the inclusion of the new classes

The results of this study can be seen in Figure 2.3-3.

Summary	FUF-Functionality	FUF-Classes	FUF-Methods Complexity	FUF-Interactions
Feedback	High 90%	Low 27%	Medium	Medium/high 66%
Undo	Medium 40%	Low 10%	High	Medium/high 66%
Cancel	Medium 95%	Low 8%	High	Medium/high 66%
User input errors prevention/correction	Medium 36%	Low 11%	Medium	Low 6%
Wizard	Low 7%	Low 10%	Low	High 70%
User Profile	Low 8%	Medium 37%	Medium	Low 10%
Help	Low 7%	Low 6%	Low	High 68%
Use of different languages	Medium 51%	Low 10%	Medium	High 70%
Alert	Low 27%	Low 7%	Low	Medium/high 66%

Figure 2.3-3 Mean values for design impact for all FUFs considered by Juristo et al. in 2007

With the results of these experiments, the authors provide initial proof of the importance of dealing with these types of usability features as early as the design phase of the development process, due to the demonstrated cost of including them a posteriori.

In the two works presented above, the authors show that there is a clear relationship between certain usability factors and the underlying software architecture, and as such, it's one that must be carefully regarded during software development.

## 2.4 Architectural patterns

The studies grouped in this section propose solutions to addressing usability concerns at design time during application development. The solutions include general, text-based architectural recommendations, usability-specific architectural patterns, assessment techniques to determine the needs of an architecture in regards to usability and matching existing design patterns to existing usability needs.

In the following sections detail the five most representative studies, as selected by the systematic literature review, from least to most recent. This will give the reader a perspective of the work carried out in this field thus far by the various research groups that have attempted to or are currently tackling the matter, the advances that have been made and the present shortcomings to address.

### 2.4.1 Linking usability to software architecture patterns through general scenarios

*Len Bass and Bonnie John. Journal of Systems and Software. 2003 [3]*

In this 2002 work, the authors identify a set of usability scenarios that appear to have architectural implications, determine their potential usability benefits and propose software architectural (SA) patterns to help users realize those benefits.

The usability scenarios selected by Bass and John in this work resulted from literature searches, discussions with colleagues and their own personal experience. All of the 27 scenarios that were chosen were described as architecturally significant, in that a solution for them would always affect the software architecture.

The selected usability scenarios are presented in Table 2.4-1.

Table 2.4-1 General Usability Scenarios, Bass, John et al. 2002

Aggregating data	Aggregating commands	Canceling commands
Using applications concurrently	Checking for correctness	Maintaining device independence
Evaluating the system	Recovering from failure	Retrieving forgotten passwords
Providing good help	Reusing information	Supporting international use
Leveraging human knowledge	Modifying interfaces	Supporting multiple activities
Navigating within a single view	Observing system state	Working at the user's pace
Predicting task duration	Supporting comprehensive searching	Supporting undo
Working in unfamiliar context	Verifying resources	Operating consistently across views
Making views accessible	Supporting visualization	Supporting personalization

For each of the selected scenarios, the authors generate a solution in the form of a high-level software architectural pattern and present it as a possible way to implement said scenarios.

The SA pattern that is provided as a solution for the Cancelling Commands usability scenario in this work, is shown in Figure 2.4-1.

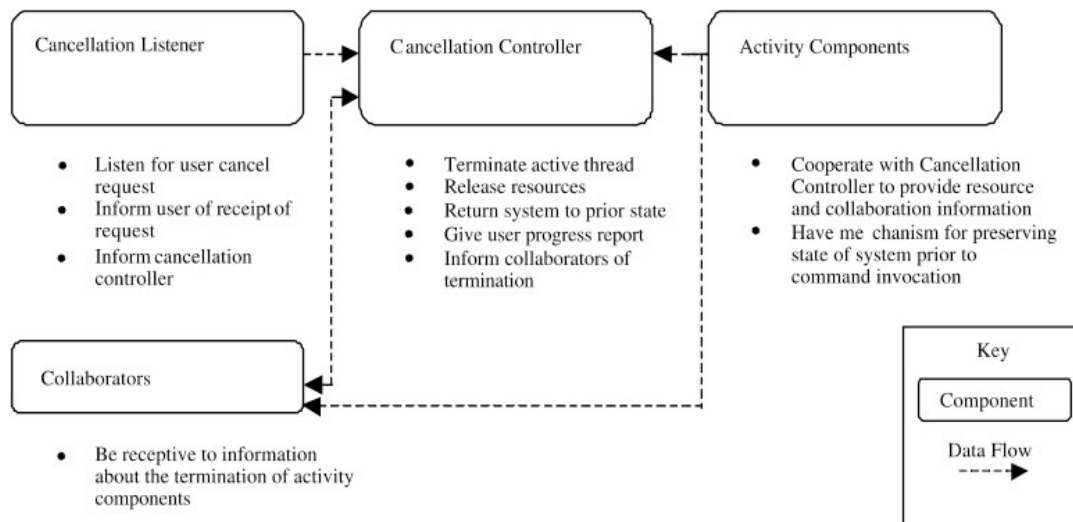


Figure 2.4-1 SA pattern proposed for the Canceling Commands scenario. Bass, John et al. 2002

Each of the proposed scenarios is solved in a similar manner, a high-level depiction of the proposed architectural pattern followed by a textual description of it. The full set of patterns is presented in a CMU/SEI Technical Report titled “Achieving Usability Through Software Architecture”, by Len Bass, Bonnie E. John and Jesse Kates.

Aside from providing an architectural solution to each of the twenty seven scenarios, they were also classified in two ways

- From the point of view of the usability benefits that could be reaped by the user from its implementation
- From the point of view of classifying the architectural patterns proposed as solutions for each of them

This two-way classification yielded a usability benefit vs. architectural tactic matrix, termed ‘benefit/tactic matrix’ for short. The goal of this matrix is to provide help for the user in understanding the benefits of each scenario and also in evaluating and designing their system’s architecture.

This benefit/tactic matrix was applied for a couple of scenarios (‘working in an unfamiliar context’ and ‘multiple languages’) to a large commercial information system, leading to the examination of its existing designs and to ultimately solving difficulties present in the system for both scenarios.

As future work in this study, Bass and John propose, among other goals, quantifying the value of the potential benefits of the proposed scenarios to the users and fleshing out the architectural solutions into usable patterns like those of Gamma et al. [37].

It was later reported by the authors in [4] that only the Canceling Commands was ever fully fleshed out in this manner.

## 2.4.2 A Software Architectural View of Usability Patterns

*Xavier Ferre, Natalia Juristo, Ana Maria Moreno, Maria-Isabel Sanchez-Segura. Conference Proceedings of INTERACT 2003 [19]*

In this study, the authors identify twenty usability patterns that, when present in a system, improve its usability. For each of these patterns and through the inductive process summarized below, the authors produce a possible design solution for incorporating them into the architecture of software applications.

In order to identify the usability patterns for which to propose architectural solutions, the authors took a top-down approach, decomposing well-known usability attributes as defined in literature by Nielsen and others into what the authors have termed usability properties, and finally further down into usability patterns. Figure 2.4-2 shows a partial view of this decomposition as proposed by the authors.

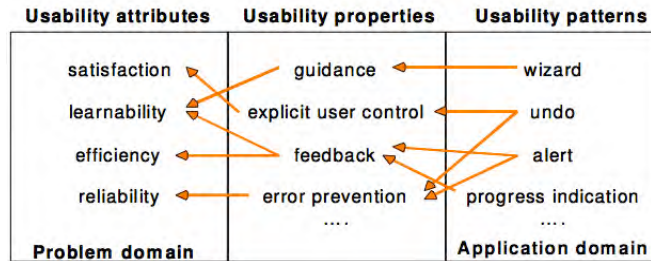


Figure 2.4-2 Relationships between usability attributes, properties and patterns. Ferré et al. 2003

The purpose of producing these usability patterns was to bring the high-level usability attributes down to a level of abstraction for which specific architectural solutions could be proposed. The full list of usability patterns obtained in this work is shown in Table 2.4-2.

Table 2.4-2 Final list of usability patterns produced by Ferré et al. 2003

Different languages	Different access methods
Alert	Status indication
Shortcuts	Form/field validations
Undo	Context-sensitive help
Wizard	Standard help
Tour	Workflow model
History logging	Provision of views
User profile	Cancel
Multi-tasking	Commands aggregation
Action for multiple objects	Reuse information

For each of these usability patterns, an architectural pattern was identified inductively: In several laboratory studies, the authors asked the software designers to build their systems without considering any of the usability patterns. Once designed, designers were asked to modify their systems in order to include the usability patterns. Then, from the modifications made by each of the teams, the authors abstracted an architectural pattern for each pattern.

Though the authors report not having validated the feasibility of the resulting patterns, they state the importance of applying them in real software developments for this purpose.

The resulting architectural patterns that were abstracted conformed to a template that contained the following information:

- Pattern name
- Problem: When to apply the pattern and in which context
- Solution: A high-level, generic architectural diagram (see Figure 2.4-3) and a description of the participants involved in the interaction.
- Usability benefits: The specific usability aspects that can be improved by this pattern
- Usability rationale: The impact of the present pattern in final usability
- Consequences: Impact of the pattern in other quality attributes of the system
- Related patterns
- Implementation of the pattern in OO: A textual description of how the generic diagram could be translated into an object-oriented design.
- Example: One of the designs used to produced the present pattern

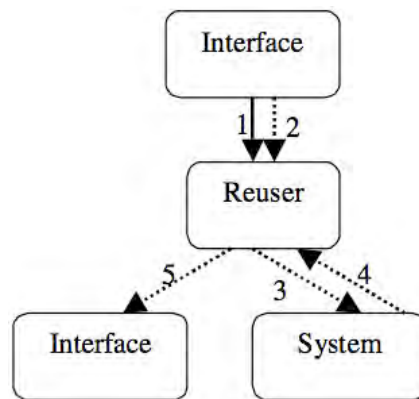


Figure 2.4-3 Architectural pattern proposed for the Reusing Information usability pattern. Ferré et al. 2003

Though no formal validation was ever reported on this particular set of patterns, this study represents a step forward in ratifying the positive relationship between architectural decisions and software usability.

### 2.4.3 Bringing usability concerns to the Design of Software architecture

*Bonnie John, Len Bass, Maria-Isabel Sanchez-Segura, Rob J. Adams. Lecture Notes on Computer Science. 2005 [30]*

In this work, the authors introduce what they have termed Usability Supporting Architectural Patterns, or USAPs. Each USAP describes a usability concern, provides a set of responsibilities that must be fulfilled to satisfy the forces involved in said concern, and describes an MVC-based sample solution for it.

This work follows the same research line as the 2002 work presented above, with one major difference: the consideration of the aforementioned forces.

The authors point out that in their previous approaches to this problem, including their own previous approach, no good traceability was provided between the solutions provided and the specific aspects of the usability scenarios being addressed. This disconnect was proposed to be remedied by the concept of forces [1].

Forces are defined as emanating from different parts of an organization unto a system of people and machines and causing a particular task to be undertaken. When pertaining to software architecture, the authors focus on forces coming from “the task the software is designed to accomplish, the environment in which it exists and the desires and capabilities of humans using the software”. The authors propose combining these forces in order to produce the usability problem to address, along with a set of responsibilities that must be present in the software design in order to solve the problem.

Each USAP, as proposed by the authors, is meant to conform to a table template containing the following main elements:

- **Usability context:** the name of the pattern, the situation, or description of the pattern from the point of view of the user, and any potential usability benefits
- **Problem:** The forces that motivate the usability situation, separated as exerted by: the environment and the task, human desires and capabilities and the state of the software.
- **General Solution:** Textual description of general responsibilities the must be present in the a system purporting to solve the problem. These general responsibilities as well as the problem they address were deduced by the authors after examining prior research in usability and software architecture.



- **Specific Solution:** The usability-supporting architectural patterns, derived by the authors from the general responsibilities described above, as well as from the forces exerted prior design decisions. These patterns are made up of component and sequence diagrams along with textual descriptions of the rationale behind the chosen responsibility allocation, and the forces exerted by the choice of over-arching architecture (J2EE-MVC). These patterns

The authors present one full example USAP for the Cancellation scenario. Figures Figure 2.4-4 and Figure 2.4-5 show the proposed component diagram for this pattern as well as one of the proposed sequence diagram for the solution, respectively.

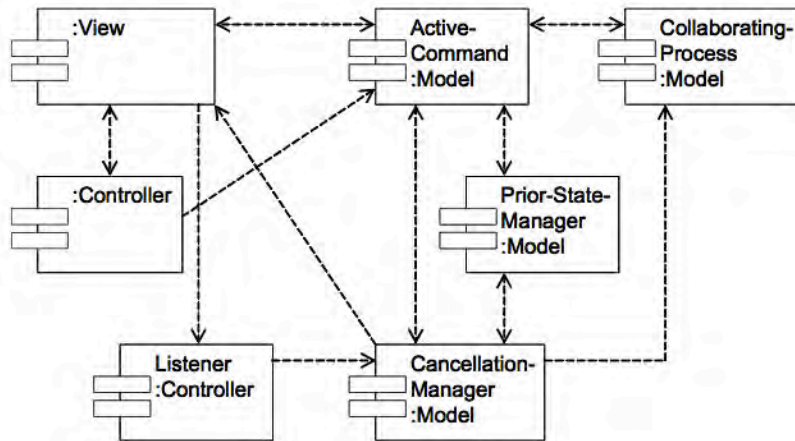


Figure 2.4-4 Component diagram for the SA pattern of the Cancellation scenario. Bass, John et al. 2005

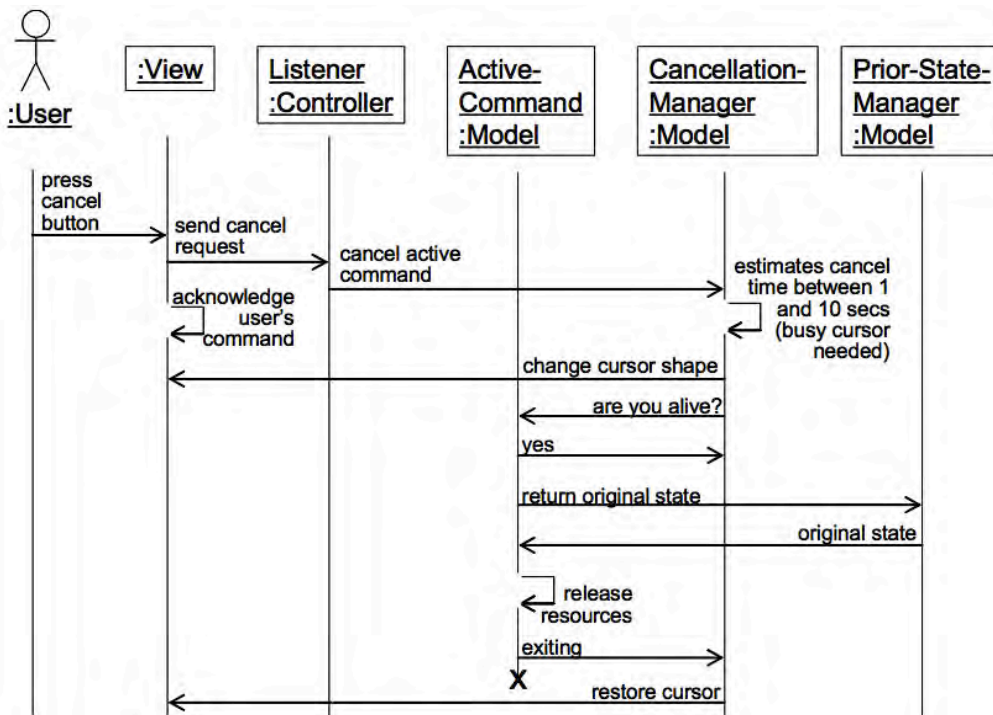


Figure 2.4-5 Sequence diagram for the SA pattern of the Cancellation scenario. Bass, John et al. 2005

The ultimate goal of this work was to construct a collection of USAPs formatted as described in this study, to be used by architectural and design teams in the development of projects

The USAPs were presented to professional audiences receiving encouraging feedback and some were also validated in practice, but in both cases this was done prior to the inclusion of forces as proposed in this work.

#### 2.4.4 Reconciling usability and interactive system architecture using patterns

*Seffah, A; Mohamed, T; Habieb-Mammar, H; Abran, A. JSS. 2008 [39]*

In this study the authors identify and model specific scenarios that illustrate how internal software components, referred to as ‘invisible’ components may affect a system’s usability. For each of the proposed scenarios, an existing or improved software design pattern is suggested as a potential solution to the scenario. These scenario-pattern pairs are ultimately documented and their application within a MVC architectural model is detailed.

In this work, the authors chose the following set scenarios, extracted from literature reviews and day-to-day experiences:

- Time consuming functionalities
- Updating the interface when the model changes its state
- Performing multiple functionalities using a single control
- Invisible entities keep the user informed
- Providing error diagnostics when features crash
- Technical constraints on dynamic interface behavior

Following their identification of the scenarios, the authors propose existing design and interaction (HCI) patterns as solutions for them. For example. The “Working Data Visualization” design pattern is proposed as a solution for the “Updating the interface when the model changes its state” scenario. Similarly, the “Progress Indicator” interaction pattern is proposed as solving the “Time consuming interactions” scenario.

Each pattern is described extensively, providing its name, the problem to solve, the context in which it exists, the forces involved, a textual solution, the resulting context after applying the pattern and the ultimate effects on the resulting usability.

Though this process is repeated for a limited number of patterns, the long-term goal of this work is said to be proposing a general theoretical framework for identifying scenarios such as the ones mentioned above and defining patterns for developers that need to be used in order to solve them.

In working towards this goal, the authors propose a model of the cause-effect relationships between software elements and usability, identifying the most probably types of cross-relationships between usability and software architecture (bold links in Figure 2.4-6).

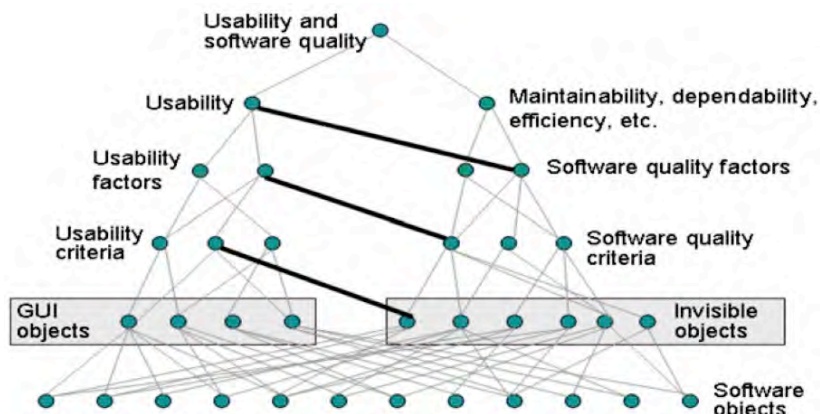


Figure 2.4-6 Most probable types of relationships between usability and architecture. Seffah et al. 2008



The authors suggest that the proposed model is useful in helping developers understand where to look for relationships between architecture and usability, though stating that one of their future objectives is validating their results.

#### 2.4.5 A Responsibility-Based Pattern Language for Usability-Supporting Architectural Patterns

*Bonnie John, Len Bass, Elspeth Golden, Pia Stoll. Conference Proceedings. ACM SIGCHI EICS 2009 [5]*

Following their research on Usability-Supporting Architectural Patterns (USAPs) described in section 2.4.2, the authors looked to test them in industry. Though the authors report having only had success in creating one full exemplar, the Cancellation pattern, (due to difficulties encountered in creating and maintaining their originally intended catalogue of about two dozen patterns) this particular pattern was proven to improve the quality of architecture designs for supporting this usability characteristic. To further study and validate the rest of the partial USAPs in the catalogue a select few were initially chosen by the company involved in this study, ABB, to include in their current product line. The authors were quickly made aware of the multiple challenges posed by the use of the patterns as they were. This led to the alteration of the USAP structure as well as to the proposal of a pattern language based on software responsibilities, alongside a web-based tool for evaluating an architecture with respect to those patterns.

During the beginning stages of this industry-set experiment, the authors, together with an architecture team of ABBs, completed and reformatted the USAPs (down to the UML sequence diagrams as the Cancellation pattern had been) for two specific usability scenarios: User Profiles and the Warning/status/alert feedback.

Upon presenting these new USAPs to a different architecture team, the authors realized that the general response was a resistance to the use of a UML-based solution, particularly one based on an over-arching pattern such as MVC. The architects in question felt pressured into redesigning their systems to comply with the patterns, instead of perceiving them as helpful tools for including the usability characteristics in question.

The architects were also put off by the nearly hundreds of responsibilities expected of the system, and further wondered whether three or more of these large patterns could be used together within a single architecture design.

Such feedback led the authors to remove the UML diagrams altogether from their patterns, and replace them with textual descriptions of implementation details. These descriptions explained the structure and behavior of the solution, without imposing a particular architecture.

Furthermore, the many responsibilities contained in each USAP were grouped into categories, leading to a two-level structure with emphasis on the hierarchical relationships between the structure's elements. Furthermore, the concepts of End-user USAP and Foundational USAP were proposed, leading to the creation of a pattern language, based on Alexander's principles [1] Figure 2.4-7 depicts part of the structure of this language.

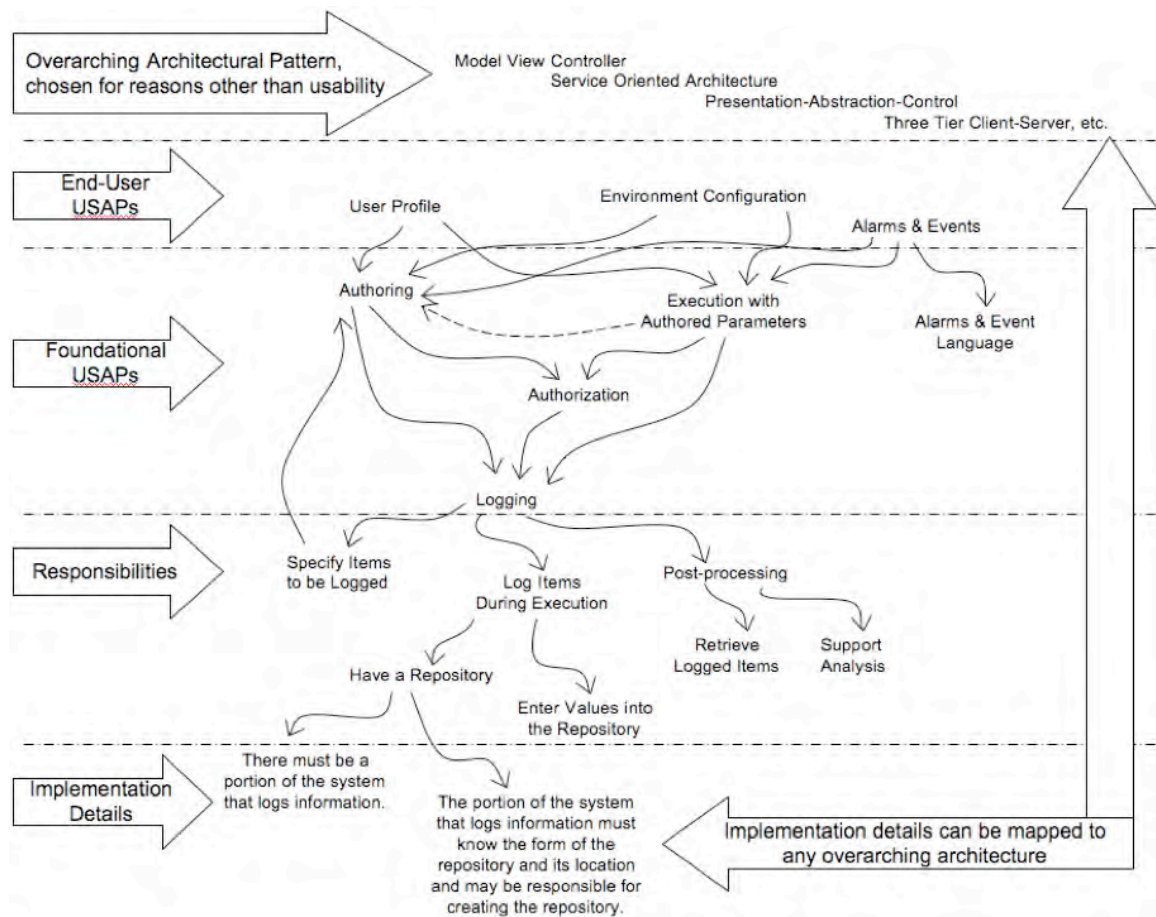


Figure 2.4-7 Scales of proposed pattern language. Bass, John et al. 2009

By expressing the USAPs through this pattern language, the authors were able to reuse 84% of the responsibilities across all patterns, resulting in a 63% reduction of the number of responsibilities that had to be presented to the software architects. This fact, together with a proposed checklist-based delivery tool which automated the tailoring of patterns for a project's particular needs, made architects more welcoming to the USAPs design suggestions, with some reports of highly increased productivity stemming from its use.

While the authors acknowledge that actual validation of results still remains to be carried out, they are optimistic about the viability of their proposed pattern language judging by the results of this experiment.

## 2.5 Systematic literature review results

The past two decades have seen extensive amounts of research carried out in regards to understanding and quantifying the strong relationship that exists between software architecture and usability. These results highlight the importance of said relationship and the need to address usability concerns from a software architecture perspective. Multiple approaches have been explored for addressing said concerns, mainly proposing diverse forms of architectural frameworks, guidelines and patterns in order to include usability into software systems correctly and effectively.

While the results obtained thus far are encouraging, there is still work to be done in this field. For instance, most of the aforementioned works are substantially backed up by prior research proving the usability features the authors chose to address are in accordance to usability principles that have been proposed in the HCI field. Furthermore, no empirical validation was

performed in any of these works, save for the three case studies carried out by Folmer, et al. (2005) in [20] and one, under way at publication time, by Ferré, et al. (2003) in [19].

Furthermore, the usability issues addressed in existing works as starting points are identified mostly by heuristic-based approaches. Ideally, the usability concerns to consider when proposing architectural and/or design patterns should be relevant from an HCI perspective and have proven implications on software architecture and/or design.

Most of the previous works deal with solutions at a high-level architectural level, which are not adequately validated. While architectural patterns can be very useful in depicting how a system should behave as a whole, our work explores the option of lower-level design solutions being more effective in detailing the responsibilities of its components.

In addition, none of the works studied provide any means of traceability between their proposed solutions and software requirements, which is of utmost importance for validation and maintenance purposes.

Therefore, we are presented with an open research problem related to providing users with efficient design and implementation artifacts to incorporate usability into a software system, and we intend to address it within the scope of this doctoral thesis.

A summarized view of the aforementioned shortcomings of existing research results is presented in Table 2.5-1, alongside our proposed solution.

Table 2.5-1 Summarized view of systematic literature review results

#	Authors	Study Title	Working set of usability features or scenarios		Characteristics of proposed design solutions (if applicable)		Full empirical validation of results	Traceability between proposed design solutions and software requirements
			Source	Focus on those with proven impact on design	Abstraction level	Usage of existing patterns vs. new solutions		
5	Folmer, et al.	Software architecture analysis of usability	Literature searches	No	-	-	Yes	None
15	Juristo, et al.	Analyzing the impact of usability on software design	Research results	Yes	-	-	Yes	None
1	Bass, et al.	Linking usability to software architecture patterns through general scenarios	Literature searches and heuristic approach	No	High	New	Partial	None
23	Ferre, et al.	A software architectural view of usability patterns	Research results	Yes	High	New	No	None
7	Bass, et al.	Bringing usability concerns to the design of software architecture	Literature searches and heuristic approach	No	High and low (partial) <sup>5</sup>	New and existing	No	None
19	Seffah et al.	Reconciling usability and interactive system architecture using patterns	Heuristic approach	No	High	Existing	No	None
28	Bass et al.	A responsibility-based pattern language for usability-supporting architectural patterns	Heuristic approach	No	High	New and existing	No	None
PRESENT WORK			Research results	Yes	High and low	New and existing	Yes	Yes

<sup>5</sup> Only for one of the more than two dozen scenarios was a low-level solution ever fleshed out

## CHAPTER 3. HYPOTHESIS AND APPROACH TO SOLUTION

### 3.1 Introduction

As explained in Chapter 2, developing software that properly includes usability characteristics with impact on application logic is not a trivial task. Given this fact, the objective of this work is to provide a process to help developers include such usability characteristics into their applications throughout their development life cycle. This is accomplished with the aid of a set of Software usability guidelines, also proposed in this work.

In order to address the shortcomings of existing solutions, the proposed guidelines are to:

1. Focus only on usability characteristics with proven impact on software design.
2. Be properly validated empirically.
3. Provide software design solutions that reach a low level of abstraction.
4. Be traceable to requirements.

### 3.2 Hypothesis

The hypothesis for this work states that:

“Applying the proposed usability-oriented software development process will:

- Reduce development **time** of the usability-related functionalities and, as a consequence, of the project over all,
- improve the **quality** of resulting software designs,
- facilitate the inclusion of functional usability features into software projects by reducing the **perceived complexity** of usability features by developers,

over applying the process partially *and* over not applying it altogether.”

The corresponding null hypothesis is the following:

“When applying the proposed usability-oriented software development process,

- development **time** is not reduced,
- **quality** of the resulting designs is not improved,
- **perceived complexity** of usability features is not reduced,

over applying the process partially *and* over not applying it altogether.”

### 3.3 Approximation to solution

In order to validate these hypotheses, we propose a process for including usability characteristics with impact on software design into software applications, which we have termed the Usability-oriented Software Development Process. This process is supported by a set of Software Usability Guidelines for Software Development that we propose for addressing these high-impact usability characteristics. These characteristics are described in section 3.3.1, followed by an overview of the proposed process in section 3.3.2, and an outline of the proposed guidelines in section 3.3.3.

#### 3.3.1 Functional Usability Features

This work focuses on eleven usability characteristics whose effects go beyond the user interface. In [31] Juristo, Moreno and Sanchez-Segura show how these characteristics have a considerable impact on the software logic (in terms of additional classes, methods and relationships that must be implemented) when they are included in an application. Termed Functional Usability Features by the authors, they are grounded on solid HCI principles as shown in [32], and our proposed process focuses on helping developers properly include them into their software application when needed.

Table 3.3-1 Functional Usability Features, originally proposed in [32]

FUNCTIONAL USABILITY FEATURE	HCI Authors' Label	GOAL
System Status Feedback	Modeless Feedback Area [17] Status Display [42]	To inform users about the internal status of the system
Progress Feedback (a.k.a Long Action Feedback)	Progress Indicator [42] Show Computer is Thinking [11] Time to Do Something Else [11] Progress [49] Modeless Feedback Area [17] Let Users Know What is Going On [7]	Informs the users that the system is processing an action that will take some time to complete
Warning	Think Twice [11] Warning [49]	To inform users of any action with important consequences
Undo	Multi-level Undo [42] Undo [49] Global Undo [36] Allow undo [11] Go Back One Step [42] Object-Specific Undo [36]	To undo system actions at several levels, either globally or over an individual object
Abort	Go Back One Step [42] Emergency Exit [11] Cancellability [42]	To cancel the execution of an action or the whole application
Step-by Step Execution	Step-by-Step [42] Wizard [49] [42] Go Back One Step [42] Go Back to a Safe Place [42]	To help users do tasks that require different steps with user input and correct such input
Preferences	User Preferences [42] Preferences [49]	To record each user's options for using system functions
Personal Object Space	Personal Object Space [42]	To record each user's options for using the system interface
Favorites	Favorites [49] Bookmarks [42]	To record certain places of interest for the user
Multilevel Help	Multilevel Help [42]	To provide different help levels for different users
Command Aggregation	Composed Command [42] Macros [42]	To express possible actions to be taken with the software through commands that can be built from smaller parts

The original set of functional usability features proposed in [32] is made up of fifteen features. The focus set in this work is made up of only eleven, as they are the result of merging and reconfiguring some of the originally proposed features.

### 3.3.2 Usability-oriented software development process overview

The usability-oriented software development process proposes activities to be carried out by software developers during different phases of development of their applications. Regardless of specific project life cycle or development method choices, the proposed activities are to be applied during the analysis and design phases of a project (or a project's iteration, sprint, etc.). Figure 3.3-1 shows a graphic representation of the proposed process.

The first part of the proposed process takes place during the Elicitation and Analysis phase of the project (or project iteration, sprint or similar). During this phase, the development team, most typically the analyst(s) will carry out activities inherent to the development process through which the project is being built. Figure 3.3-1 refers to these activities as "Traditional requirement elicitation and analysis activities". These activities typically include interaction with the project stakeholders (end-users, clients, etc.) and other software analysis activities with the goal of gathering and documenting the user's needs for the project. This could be done as a Software Requirements Specification (SRS), a collection of user stories, a set of expanded use-cases, user interface prototypes, etc.

Together with these traditional activities, the analyst(s) would also carry out those proposed by our process, represented in Figure 3.3-1 as the "Requirement elicitation and analysis activities for usability" sub-process. These activities gather and document requirements related to the functional usability features with the help of the proposed Usability Guideline, specifically its analysis artifacts.

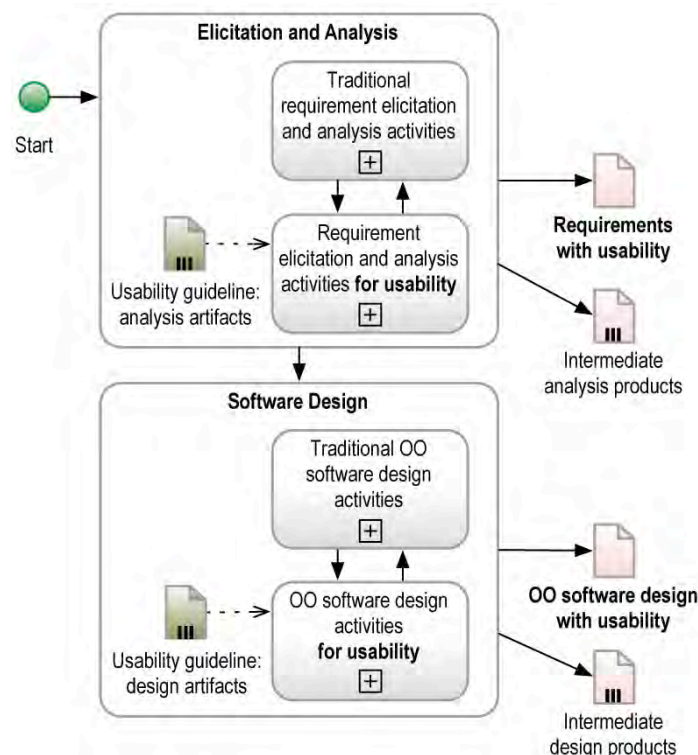


Figure 3.3-1 Usability-oriented software development process overview in BPMN [50].

How these traditional activities will be performed together with the proposed activities will depend on multiple factors inherent to the project being developed. Such factors may be the type and size of the project, the amount of requirements, the problem domain, the team's experience, etc. In some cases it may prove useful to first perform all the traditional requirements elicitation activities and then follow with all the usability-related ones, while in others the team might go back and forth between the two until a final requirements document

(including usability) is produced. This document, represented in Figure 3.3-1 as the “Requirements with usability”, is the final product of this stage (represented in bold letters).

Other intermediate and/or optional analysis products are also part of the output from this Elicitation and Analysis phase, represented by the multi-document icon labeled “Intermediate analysis products”. One of these products is a use case model for the project, produced with the help of the analysis artifacts of the Usability Guideline for teams (optionally, for teams who model use cases). Another intermediate product of this stage is a list of System Responsibilities for Usability, which are a representation of the user’s needs regarding usability expressed as general functionalities that should be present in the resulting system.

The Requirements with usability (bold text) and the intermediate products (plain text) for this phase are explained in detail in Chapter 4.

Our proposed process also provides for a similar scheme to be applied during the design phase of a project or iteration. During this phase, traditional software design activities are carried out, along with the design activities proposed by our process. As in the case of analysis, the way in which these two sets of activities will be carried out (in parallel, sequence, etc.) will depend on the characteristics of the project, team and system being developed.

The main result of the design phase is a software design document which represents the usability needs that were elicited previously as MVC-based class and sequence diagrams. This document is shown in Figure 3.3-1 as “OO software design with usability”, and is the final product for this phase, represented in bold letters.

Also during this phase, other intermediate design products that are also produced. One such product, produced before the OO designs, describes the System Responsibilities for Usability from the point of view of design, describing them in terms of software component responsibilities yet not posing constraints on a specific architecture to be used. This product is termed the High-level component responsibilities for usability and is useful as a guide on how to distribute responsibilities among components when not producing the final designs. Another intermediate product, also produced before the OO designs, is what we have termed the Low-level component responsibilities for usability, which textually express the System Responsibilities for Usability at an even lower level of abstraction, breaking them up and assigning sub-responsibilities to implementable objects and classes.

The OO software designs with usability (bold text) and the intermediate products (plain text) for this phase are also explained in detail in Chapter 4.

It is worth noting that our proposed process caters mainly to projects whose software design is object-oriented. For projects that apply a different design paradigm, our proposed process can be applied partially, specifically up to the High-level component responsibilities for usability. In such cases, developers would benefit from early tasks in the process and would carry on the software design independently. This flexibility is also present within the different activities of our proposed process, as described later in Chapter 4, allowing for it to be customized to the development team’s needs.

### 3.3.3 Usability Guidelines for Software Development

Each of the aforementioned tasks that make up the full usability-oriented software development process proposed in this work, relies on a tangible product to perform their duties. This product is the Software Usability Guideline, shown in Figure 3.3-2, which is a set of analysis and design artifacts proposed to support the inclusion of the functional usability features presented earlier in section 3.3.1 into software applications.

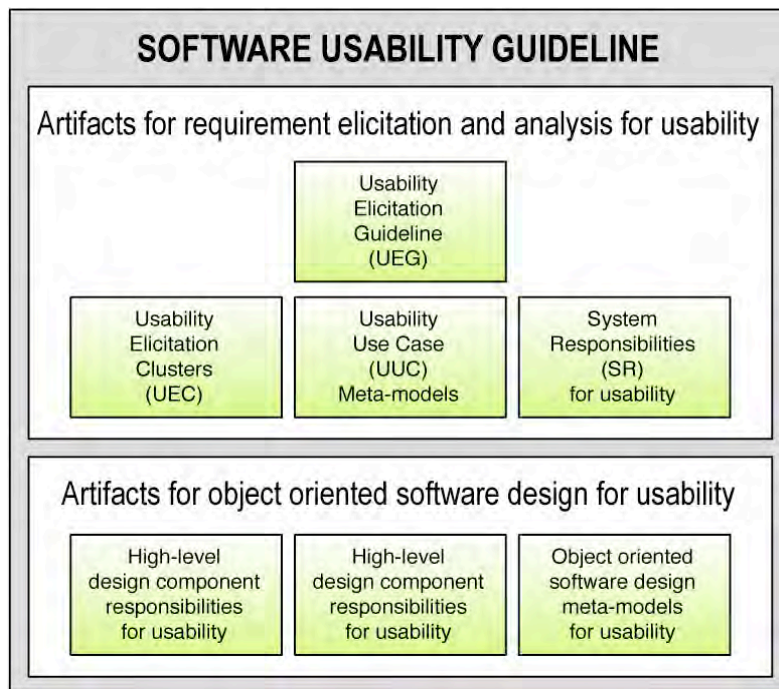


Figure 3.3-2 Structure of the Software Usability Guideline.

One Software Usability Guideline is proposed for each of the functional usability features. Chapter 5 describes each of these guidelines for all of the features addressed in this work.

Each Software usability guideline is comprised of nine artifacts, each of which is to be used during a single task. Their structure, purpose and function within it are described in depth the following section.

- The Usability Elicitation Guideline is an existing contribution by [32], extended for this work, whose aim is to help analysts in eliciting usability requirements
- The Usability Elicitation Clusters are a graphic representation of the Usability Elicitation Guideline that help analysts understand the flow of the requirements discussion items
- The Usability Use Cases Meta-model is a use case representation of the usability needs covered by the Usability Elicitation Guideline to help developers include them in their use case models
- The System Responsibilities are the main functionalities that the system should accomplish in order to potentially fulfill all of what has been elicited with the Usability Elicitation Guideline.
- The High-level design component responsibilities for usability describe the System Responsibilities at a lower abstraction level, that of high-level design components.
- The Low-level design component responsibilities for usability describe the System Responsibilities at a lower abstraction level, that of design components for a Model-View-Controller (MVC ) architecture
- Finally, the Software Design Meta-models are the graphic representation, as class and sequence diagrams, of the low-level design component responsibilities for usability.



## **CHAPTER 4. USABILITY-ORIENTED SOFTWARE DEVELOPMENT PROCESS**

### **4.1 Introduction**

This chapter provides a detailed description of our proposed process for helping software developers include usability into their applications through the use of software usability guidelines, also proposed in this work.

Section 4.2 presents an overview of the usability-oriented software development process, followed by section 4.3 describes each phase of the process in detail and explains how the proposed software usability guidelines are used for their application. Finally, section 4.4 describes a preliminary software tool for automating the proposed process.

### **4.2 Process overview**

Figure 4.2-1 presents a more detailed view of the “Requirements elicitation and analysis activities for usability” and the “OO software design activities for usability” sub-processes, shown collapsed earlier in Chapter 3.

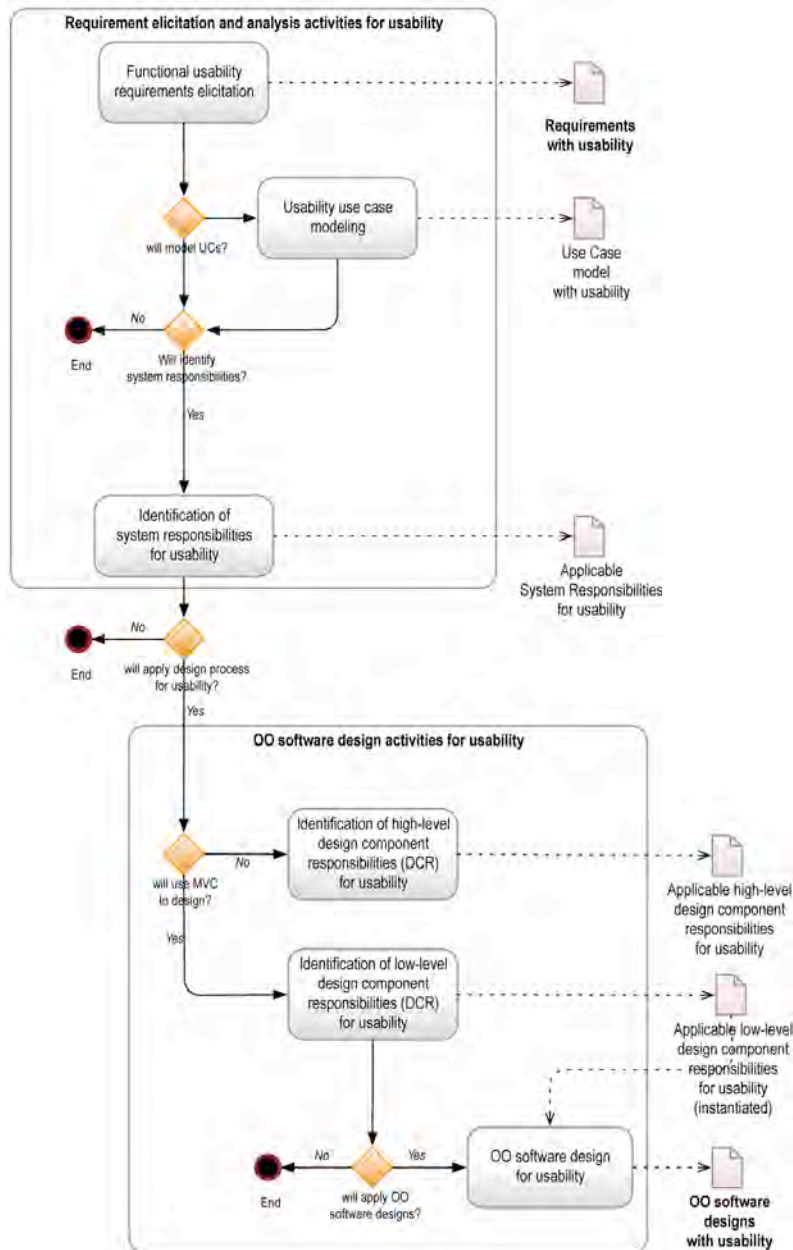


Figure 4.2-1 Usability-oriented software development process overview

The first task of the “Requirements elicitation and analysis activities for usability” sub-process is the “Functional usability requirements elicitation”. This task enhances the original requirements document (elicited through traditional means, as explained earlier in this section) with the end user’s needs regarding usability (in the case of a SRS, these needs would be expressed as functional requirements). A second, non-mandatory task in this sub-process is the “Usability use case modeling” which enhances the Use Case model for the application being developed, if one exists, with usability use cases.

Once these initial tasks are completed, the proposed process can either end, in which case its final outputs will have been the “Requirements with usability” and, optionally, the “Use case model with usability”, or it can continue, depending on the developer team’s needs. If the process is to continue, the next task is the “Identification of system responsibilities for usability”. This task produces a list of usability-related system responsibilities that the application under development should ultimately perform. This ensures that the resulting application will conform to the usability-related functionality introduced into the requirements during the first two tasks.

After these system responsibilities for usability have been identified, the development team can either stop applying the proposed process or continue applying it through to the design phase of their project or iteration. If continuing onto design with our proposed process, the next task is the “identification of high-level design component responsibilities for usability”. This task takes the system responsibilities for usability as input and produces an intermediate output document containing a list of high-level design components proposed to carry out those system responsibilities. This document also explains how these high-level components are to carry out those system responsibilities by expressing them into more finely-grained responsibilities to be distributed among the proposed components.

After these high-level design component responsibilities for usability have been identified, the proposed process can again stop, keeping the output produced so far, or continue on. If it is to continue, the next task is the “identification of low-level design component responsibilities for usability”. This task takes the earlier high-level design component responsibilities as input and further divides and distributes them, this time among proposed objects and classes, most of which are directly implementable. Furthermore, it specifies the required mechanics of the interactions between those objects and classes in order to attain the corresponding higher-level responsibility.

Finally, once the low-level design component responsibilities for usability have been identified, the process may continue on to the last task: “Object Oriented software design for usability”. This task takes the low-level design component responsibilities for usability as input to help developers produce their final designs including usability, namely the “OO software designs with usability”, which is the final product of the proposed process when applied in full.

### 4.3 Process detail

This section describes each of the six tasks that make up the two main sub-processes of the proposed process: the Requirements elicitation and analysis for usability, and the Object oriented software design for usability. They are described below in sections 4.3.1 and 4.3.2 respectively.

Every task in the proposed process uses one or more artifacts of the proposed Usability Guideline. Each of these artifacts is described also in the following sections.

#### 4.3.1 Requirements elicitation and analysis for usability

This section covers the first three tasks of the proposed process, specifically those that make up the “Requirements elicitation and analysis for usability” sub-process, first shown in Figure 4.2-1 as part of the full Elicitation and Analysis phase of a project. Figure 4.3-1 below shows the tasks involved in this sub-process, now also depicting the usability guideline artifacts involved (dark gray).

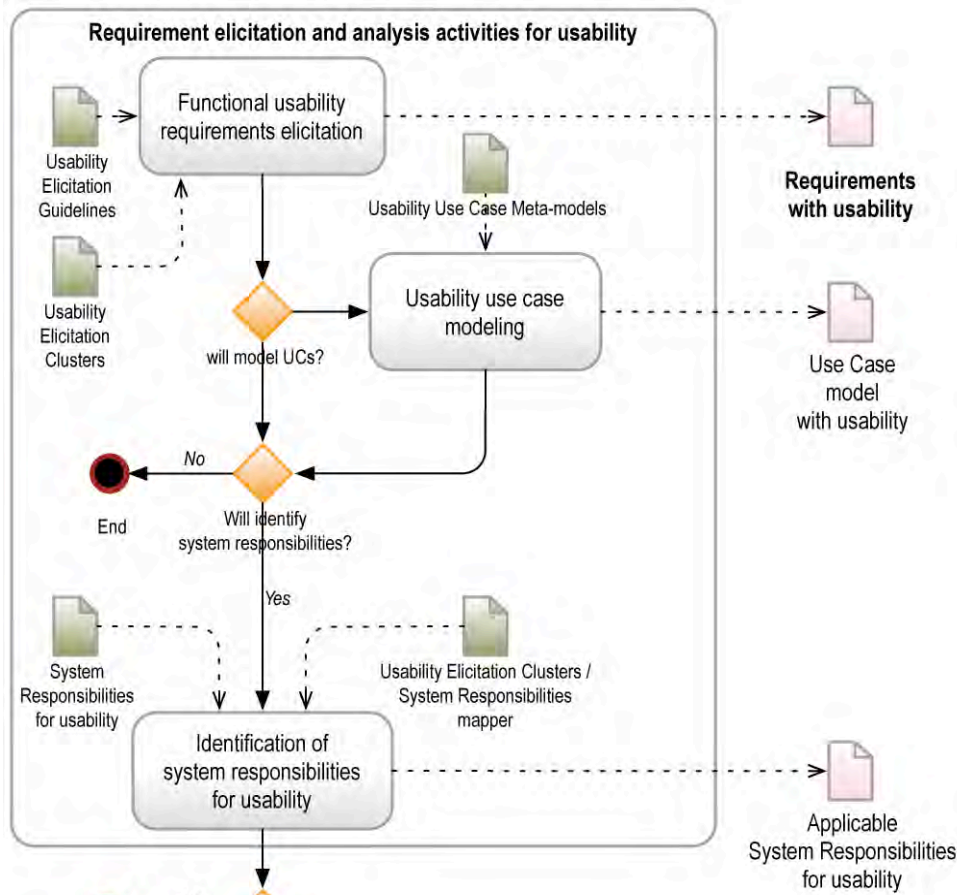


Figure 4.3-1 Requirements elicitation and analysis for usability process with the artifacts of the usability guideline for software development

#### 4.3.1.1 Functional usability requirements elicitation

This task enhances the functional requirements of a project and enhances them with usability requirements related, as elicited from the relevant stakeholders.

The next section describes the guideline artifacts that are involved in this task, followed by a description of how they support its application.

##### 4.3.1.1.1 Artifacts involved

Two guideline artifacts are involved in this task, namely the Usability Elicitation Guidelines and the Usability Elicitation Clusters, described in-depth in the following two sections.

##### 4.3.1.1.1.1 Usability Elicitation Guideline

The Usability Elicitation Guideline was originally proposed by [32] to provide help to development teams to correctly elicit all aspects related to a single functional usability feature. For a concrete example of this artifact, section 5.2.1.1 of Chapter 5 shows the full Usability Elicitation Guideline for the Undo feature.

The original Usability Elicitation Guideline had the structure shown below in Table 4.3-1. They were comprised of, firstly, identification information for the feature in question, a short definition of the problem being address by said feature and the context in which a software would benefit from its inclusion.

Secondly, and at the core of every Usability Elicitation Guideline, was the proposed solution. This solution consisted on a set of HCI recommendations, compiled from an extensive literature review for each Functional Usability Feature in [32], which fully describe the

Functional Usability Feature from a HCI perspective. The second part of the solution was a list of proposed discussion items. These were meant to be established with project stakeholders to determine how each of the HCI recommendations for the present Functional Usability Feature would be applicable to the system being developed, if at all. These discussion items would be grouped by HCI Recommendation (i.e. for each HCI recommendation there would be one or more items to be discussed with stakeholders).

The results of the stakeholders' discussions was then processed and incorporated into a Software Requirements Specification document. The last part of the Usability Elicitation Guideline, called the Specification Guide, proposed recommendations on how to incorporate this text into a Software Requirements Specification document.

Table 4.3-1 Original Structure of a Usability Elicitation Guideline [32]

IDENTIFICATION:	
Name: [FUF name]	
Family: [FUF Family]	
Alias: [Other names by which this FUF is also known in literature]	
PROBLEM: [the problem being addressed by this FUF]	
CONTEXT: [the context in which this mechanism is applicable]	
SOLUTION	
Usability Mechanism Elicitation Guide:	
HCI Recommendation	Issues to be discussed with stakeholders
[List of compiled HCI recommendations describing what software should accomplish in order for it to be considered to correctly include the present FUF]	[List of issues to be discussed with stakeholders, organized by HCI recommendations. These were mainly in the form of questions directed at the software users, clients and other project stakeholders about their needs regarding the present FUF and ways in which it would be included and used within the software]
Usability Mechanism Specification Guide:	
[Recommendations on how the results of the stakeholder discussions should be incorporated textually into a formal Requirements Specification document]	

Upon revisiting the existing structure of the Usability Elicitation Guideline within the scope of the usability-oriented software development process proposed in this work, some additions and modifications were made to its basic structure. These are described below and are shown in Table 4.3-2.

*New 'Intent' field*

To give a clearer view of what the Functional Usability Feature described in the Usability Elicitation Guideline does, an intent field was added, adhering slightly more closely to the pattern-like structure already followed by the Usability Elicitation Guidelines [32].

The Intent field states briefly, and right after the identification information of the feature, what the feature does, helping the reader to determine as soon as possible the goal of the present feature and decide on its relevance for their project.

*Removal of 'Usability Mechanism Specification Guide' field*

The original structure of the Usability Elicitation Guideline contained these recommendations on how to include the results of the discussions with stakeholders into a Software Requirements Specification document, down to a template of what the included text should look like. While such an aid would be valuable for systems in which a standard Software Requirements Specification document is always produced, for this work we have tried to keep the format in which the results of the elicitation process are gathered open. It is suggested that some form of requirements and/or use cases document is produced, but no single format is enforced, thus the original contents of the Usability Mechanism Specification Guides was no longer as widely applicable as it once was, and was thus removed from the structure.

#### *New 'Interrelationships' field*

At the end of the introductory information for the Usability Elicitation Guideline and right before the Solution section, a new field was added detailing the interrelationships that the present feature has with other features being addressed in this work. The reason for including this new field is to give the reader an idea of the impact that the inclusion (or, more importantly, the exclusion) of the present feature would have on other features.

Interrelationships included in the Usability Elicitation Guideline for feature 'A' are expressed in the following form:

In order to be able to fully include feature 'A' into the system, features B and E must also be considered

Almost all features relate and/or have other features as preconditions. Such interrelationships must be carefully looked at during elicitation, as to not risk excluding a feature that seems unnecessary to stakeholders, but is actually needed internally by another (included) feature.

#### *New 'Elaboration' column*

It was found that there was too large a 'gap' between what was expressed in the HCI recommendations and the discussions with stakeholders, to the point that it wasn't always clear to analysts using these guidelines how the proposed discussions derived from their corresponding HCI recommendation. The problem seemed to lie in the fact that the HCI recommendations were expressed not only at a very high level of abstraction, but also from a point of view too far removed from that of the more software-oriented discussions. To bridge this gap, an in-between column called Elaboration was added to the initial structure of the Usability Elicitation Guideline.

The Elaboration column presents a Software Engineering view of each of the HCI recommendations and it elaborates on topics not fully covered by the latter. For example, where a HCI recommendation would say 'actions with important consequences should be undoable' the Elaboration column would advise to go a little more in-depth into what 'important consequences' could mean for different systems during the stakeholders' discussion, and also the consideration of a cost/benefit ratio of providing particular actions with said feature.

The addition of this Elaboration column thus makes for an easier transition between the HCI Recommendations column and the Stakeholders Discussion column, as it also clarifies, from an engineering perspective, what is expressed within the HCI Recommendations.

#### *New 'Usage Examples' Column*

Another addition to the basic structure of the Usability Elicitation Guideline was the Usage Examples column. This column is optional to use during elicitation, and its purpose is to further clarify the real-life applicability of each HCI Recommendation, or row of the Usability Elicitation Guideline table, when needed.

Table 4.3-2 shows the resulting structure of the Usability Elicitation Guideline after the proposed additions and modifications.

Table 4.3-2 New Structure of the Usability Elicitation Guideline

IDENTIFICATION:			
Name	FUF name		
Family	FUF Family		
Aliases	Other names by which this FUF is also known in literature		
INTENT			
A short definition and the main goal of this FUF			
PROBLEM			
The problem being addressed by this FUF			
CONTEXT			
The context in which this mechanism is applicable			
FUF INTERRELATIONSHIPS			
Descriptions of interrelationships that the present feature has with other features			
Usability Elicitation Guideline			
HCI Recommendation	Elaboration	Discussions with stakeholders	Usage Examples (optional)
List of compiled HCI recommendations describing what the software should accomplish in order to correctly include the present FUF. Expressed as: [FUF ID]_HCI-[i]: [Title] [Description]	SE perspective of the HCI recommendation and further, lower-level explanations of unclear topics. Expressed as: [FUF ID]_ELAB-[j]: [Title] [Description]	List of issues to be discussed with stakeholders. One or more per HCI recommendation. Mainly in the form of questions directed at the project stakeholders about their needs regarding the present FUF and ways to include/use it within the system Expressed as list of: [FUF ID]_Q-[k]: [Text of issue]	Real-life examples of how this HCI recommendation is presented in existing systems Expressed as: [FUF ID]_EX-[l]: [Title] [Description]

A final addition to the original Usability Elicitation Guideline structure is the use of identifiers. As the guideline grew and more artifacts were developed, the need for clarity and traceability became paramount. In order to achieve both these goals, each item in the table above was assigned a unique identifier and given a defined structure, homogenous throughout the guideline, as explained below:

1. Each HCI recommendation in the first column is expressed using the following syntax: [FUF ID]\_HCI-[i]: [Title] [Description], where:
  - a. FUF ID, is a three-to-five letter abbreviation of the full name of the present feature. It can be either an acronym for the feature name (mostly in the case of multiple-word names) or its first few letters.
  - b. HCI, meaning that this is a HCI recommendation
  - c. [i], is a sequential number assigned to each HCI recommendation. There will be one HCI recommendation per row, hence there will be i rows in each Usability Elicitation Guideline.
  - d. [Title], a self-explanatory, one-line title assigned to the present HCI recommendation.
  - e. [Description], the full description of the present HCI recommendation
2. For every HCI Recommendation there is one or more Elaboration items, each of which is expressed using the following syntax: [FUF ID]\_ELAB-[j]: [Title] [Description].
  - a. FUF ID
  - b. ELAB, meaning that this is an Elaboration item
  - c. [j], is a sequential number assigned to each Elaboration item.
  - d. [Title], a self-explanatory, one-line title assigned to the present Elaboration item.
  - e. [Description], the full description of the present Elaboration item.
3. For every Elaboration item there is one or more Discussion Items to be had with stakeholders. Each issue is expressed in the following format: [FUF ID]\_Q-[k]: [Text of issue], where
  - a. FUF ID
  - b. Q, meaning that this is a Question (Issue) to be discussed.
  - c. [k], is a sequential number assigned to all issues of the present Usability Elicitation Guideline. It is used to identify each issue throughout the entire Usability Guideline, as they are referenced in other artifacts (namely the Usability Elicitation Clusters).
  - d. [Text of Issue], is the full text of the question or issue to be discussed.

4. Finally, for each HCI Recommendation there is one Usage Example, meaning there will be i usage examples per table. Each example is expressed as follows: [FUF ID]\_EX-[i]: [Title] [Description].
  - a. FUF ID
  - b. EX, meaning that this is a Usage Example.
  - c. [i], is the same sequential number used for the corresponding HCI Recommendation.
  - d. [Title], a self-explanatory, one-line title assigned to the present Example.
  - e. [Description], the full description of the present Example.

#### 4.3.1.1.1.2 Usability Elicitation Clusters

The Usability Elicitation Clusters are a visual guide to the stakeholder discussion items presented in the Usability Elicitation Guideline. It maps them graphically in a flow-chart-style diagram to provide an at-a-glance view of what needs to be discussed, providing the direction of the flow in which discussions need to take place and grouping related discussions in clusters to give a more abstract view of the elicitation process. A concrete example of this artifact is shown in section 5.2.1.2 of Chapter 5 shows for the Undo feature.

In every UEG, stakeholder discussion items are presented in linear lists. For each HCI Recommendation there will be one ore more discussion items that need to take place among stakeholders. From examining each discussion item individually, a reader is able to deduce if it is related to a previous discussion item (i.e. the output of discussion A is needed as input for discussion B). However, due to the high number of proposed discussion items for most of the UEGs and the sometimes complex relationships among them, it was found to be useful to represent these relationships graphically. This representation would make it easier for the reader to determine, for example, whether or not to follow down a path of discussion items, depending on what was previously discussed.

Aside from depicting the order in which discussions are to be held, the Discussion Clusters also group discussion items that relate to a single topic or goal that must be achieved by the system being developed.

The benefit of having these higher level clusters available is twofold: firstly, it gives the analyst(s) a more abstract view of what will be discussed, so that, for a UEG with dozens of discussion items, looking at the headers of the handful of clusters that comprise it will give them a clear idea of the topics that will be discussed, prior to reading all the discussion items in detail.

The second benefit of having the discussions grouped in clusters is that these clusters represent the core goals that need to be achieved by the system being developed. As such, they will give way to what will later become the general responsibilities to be fulfilled by the system, described in-depth later in this chapter.

Figure 4.3-2 shows the basic structure of an Elicitation Clusters. The chosen notation for the Elicitation Clusters loosely resembles that of flow chart diagrams. It's a simple and clear notation where the rectangles represent open discussions (those that give way to different types of output), the rhomboids represent the output, if any, of those open discussions and the rhombuses represent discussions of the form of binary questions: those with only two possible answers.

An arrow going from one discussion item (question), or more specifically, its answer, to another, denotes that not only does the second discussion item needs to take place after the first, but that it uses the output (answer) from the first discussion item as input. Finally, a line coming out of a rhombus (binary question) and ending in the 'ground' symbol represents the answer which brings that particular branch of the discussion to an end.



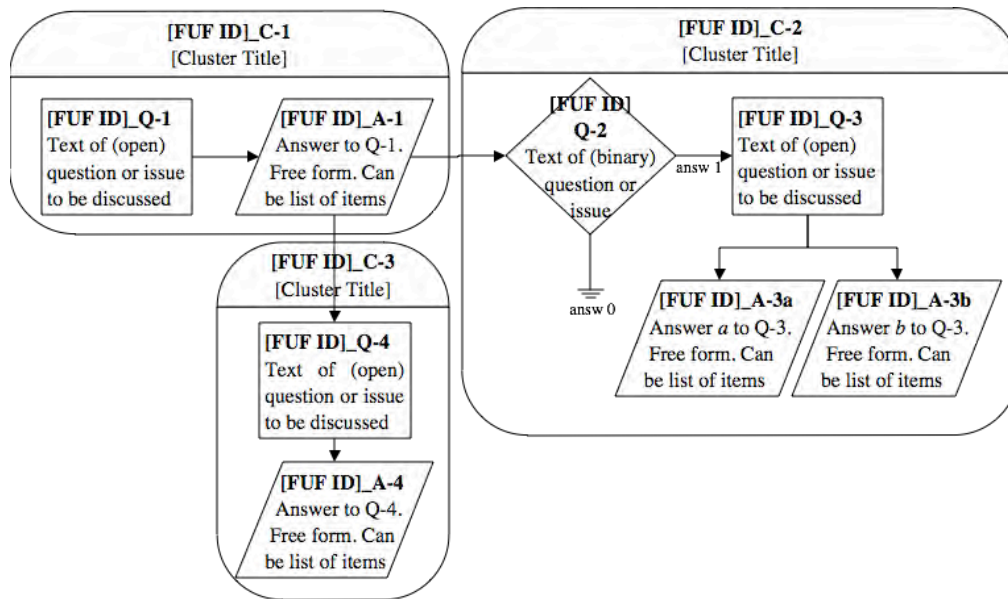


Figure 4.3-2 Structure of Elicitation Clusters

Furthermore, each discussion item is labeled with the same identifier as it is in the Usability Elicitation Guideline for traceability purposes.

Answers are assigned the same number as the question they respond to, with two notable consequences: Firstly the list of answer identifiers will be missing a number for each binary question present (in the example above there is no answer labeled A-2, as Q-2 is a binary question). This drawback is however outweighed by the clarity provided by the matching numbers between question and answer, which would otherwise be offset. Secondly, in the instances where an open question (rectangle) may have two answers, these two are labeled with the same number as the question, appending sequential letters (see A-3a and A-3b)

Finally, Discussion Clusters are identified as: *[FUF ID]\_C-[I] [Cluster Title]*, where *[FUF ID]* is the same identifier assigned to the present feature, *C* stands for Cluster, and *[I]* is a sequential number given to every cluster.

#### 4.3.1.1.2 Functional usability requirements elicitation: Task description

Once a set of functional requirements is defined (partially or totally and with more or less detail, depending on the software process and life cycled used) project stakeholders should meet to discuss the inclusion of usability requirements. In this discussion, led by the software analyst(s) and typically including end users and/or clients, the Usability Elicitation Guideline will serve as a script for what discussions need to take place in order to determine which Functional Usability Features will be required for the system being developed and the precise way in which they will be included.

In many cases, the discussion items in the Usability Elicitation Guideline, though listed sequentially due to its table structure, are not necessarily intended to be addressed in sequence. The Usability Elicitation Clusters tackle this issue by graphically mapping out the intended order in which each of these discussion items is to be addressed. Furthermore, it specifies what the expected output (or type of answer) is for each of the discussion items.

With the help of the Usability Elicitation Clusters, depending on the outcome of particular discussion items, entire ‘branches’, or series of discussion items, can be discarded and thus skipped, making the elicitation process more efficient.

By following this script, the software analyst ensures covering all the bases for each of the functional usability features, making sure that those that are expected of the system are indeed

included in the final requirements documentation as would any other functional requirements. The final output of this task is an enhanced version of the requirements document, in the sense that it includes the stakeholders' needs regarding the functional usability features, aside from the original functional requirements.

Appendix 9.1 shows an example of how this task is applied. As mentioned earlier, the proposed process can either continue on to the next task or stop here, preserving the output produced so far.

#### 4.3.1.2 Usability use case modeling

This task is optional within the “Requirement elicitation and analysis activities for usability” sub-process, applicable only when the development team makes use of use case models. If they do, this task takes a use case model (either partial or complete depending on the development life-cycle being followed) and enhances it with the appropriate usability use cases.

##### 4.3.1.2.1 Artifacts involved

The usability guideline artifact involved in this task is the Usability use case meta-model, described below.

##### 4.3.1.2.1.1 Usability Use Case Meta-model

The Usability Use Case Meta-model is in part a model in itself, in the sense that it contains specific use cases for the Functional Usability Feature that it pertains to, and in part a meta-model, as it also contains ‘template’ use cases where actual use cases will be filled in during development. A concrete example of this artifact is shown in section 5.2.1.3 of Chapter 5 shows for the Undo feature.

The main goal of the Usability Use Case Meta-model is to assist a development team in the depiction of their use case models when including a Functional Usability Feature.

Figure 4.3-3 shows the basic structure of the Usability Use Case Meta-model.

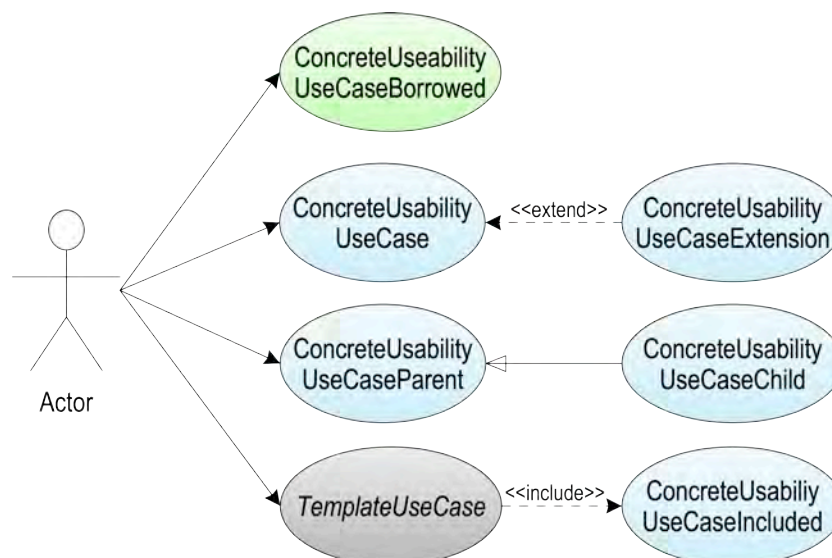


Figure 4.3-3 Structure of the Usability Use Case Meta-model

The notation used to represent the use cases contained in the Usability Use Case Meta-model is fully compliant with UML2 [22]. Some additional color coding and font changes are used to represent additional concepts, as described below.

Any single Usability Use Case Meta-model can contain any or all of the following:

1. **Concrete Usability Use Cases:** These are the use cases that embody functionality that corresponds solely to the present feature. This use case would not exist within a system's greater use case model, if said system does not include the present feature. Concrete Use Cases are represented in the current feature's assigned color (light blue in Figure 4.3-3. See Appendix 9.4 for color legend) Following UML2 notation, any of these use cases can be a Parent or Child, an Included or Including use case or an Extending or Extended use case.
2. **Concrete Usability Use Cases (Borrowed):** These are Concrete Use Cases that belong to a different feature but have some kind of connection or interaction with the use cases of the present feature. They are included in the Usability Use Case Meta-model of the present feature to emphasize said connection, but are represented using the color assigned to the feature they belong to (green in Figure 4.3-3). Whenever a Borrowed Use Case exists in the Usability Use Case Meta-model of a usability feature, the corresponding interrelationship must be noted and described in the feature Interrelationships field of the Usability Elicitation Guideline.
3. **Template Use Cases:** As their name indicates, Template use cases represent the place in the diagram where the development team must substitute the corresponding domain-specific use case. Most Functional Usability Features do not exist on their own; they are tied or triggered by other events in the system (a user executing a particular event or clicking a particular button, etc.). These actions naturally vary from system to system, but the way in which they relate to other Concrete Usability Use Cases remains a constant, so their place is reserved within the Usability Use Case Meta-model, along with the relationships it will have with the other elements present within it. These Template Use Cases are always represented in gray with italic font, to distinguish them from Concrete and Concrete Borrowed use cases.

Use cases, regardless of setting, with the exception perhaps of the simplest of systems, tend to be interrelated. For example, when choosing which use cases to implement during a particular phase or iteration of development, one must carefully determine the composition of such a set of use cases, as to not leave out any that will inevitably have to be included later for having some sort of dependency with one that is already included, or vice versa. Because of these dependencies, the Usability Use Case Meta-model in every guideline of this work is accompanied by a Usability Use Case Dependencies table like the one shown in Table 4.3-3, where the "X" identify every pair of use cases that are dependent on each another.

Table 4.3-3 Usability use case dependencies table

	[FUF_ID]_UC-1 [FUF_NAME]	[FUF_ID]_UC-2 [FUF_NAME]	...	[FUF_ID]_UC-n [FUF_NAME]
[FUF_ID]_UC-1 [FUF_NAME]	-			X
[FUF_ID]_UC-2 [FUF_NAME]	X	-		
...			-	X
[FUF_ID]_UC-n [FUF_NAME]		X		-

#### 4.3.1.2.2 Usability use case modelling: Task description

When software analysts decide to create a use case model for their system, the Usability Use Case Meta-models serve as a template for the usability requirements. Concrete use cases will be transferred directly to the project's use case model, while the so called 'template' use cases depicted in gray will be substituted for the appropriate, project-specific use case as needed.

The Use Case Meta-model contains all use cases related to the Functional Usability Feature that they belong to. However, in many cases the entirety of the feature will not be needed within a project, and only a sub-set of these use cases will be part of the final model.

Once all needed use cases in the meta-model have been included, the resulting use case model will be said to contain all the elicited usability information. At this point the proposed process can continue on to the next task or stop, keeping the output documents produced so far.

Appendix 9.1 shows an example of how this task is applied.

#### 4.3.1.3 Identification of system responsibilities for usability

The System Responsibilities that are defined for each Functional Usability Feature represent what is expected of the resulting software in order to fulfill the elicited Functional Usability Features. Analysts can identify which of these responsibilities will be needed within their system with the help of the UEC / SR mapper.

##### 4.3.1.3.1 Artifacts involved

The guideline artifacts involved in this task is the System Responsibilities list, described in-depth in the following section.

##### 4.3.1.3.1.1 System Responsibilities

As explained earlier in the elicitation and analysis phase, the stakeholder discussion items of the UEG are grouped into clusters according to their topic of discussion. These clusters represent the basis for the goals expected to be fulfilled by the system, and as such, serve as a blueprint for what the core System Responsibilities of such a system will need to be.

In order to determine the exact System Responsibility, each cluster is studied, and the main goal expected of the system is extracted. Each cluster will give way to one or more System Responsibilities, except in seldom cases where the goal of a given cluster might need to be separated into multiple System Responsibilities for clarity. A concrete example of this artifact is shown in section 5.2.1.4 of Chapter 5 shows for the Undo feature.

Table 4.3-4 shows the structure of a System Responsibility list.

Table 4.3-4 Structure of System Responsibility list

System Responsibilities List for [Name of FUF]
List of all System Responsibilities for this FUF, in the following notation: [FUF ID]_SR_[i] [Title]:[Description]

In the System Responsibilities List, each System Responsibility is labeled as follows: [FUF ID]\_SR\_[i] [Name of System Responsibility i]: [Description of System Responsibility i], where:

1. FUF ID, is the three-to-five letter identifier assigned to the Functional Usability Feature in question during elicitation.
2. SR, represents that this is a System Responsibility.
3. [i], is a sequential number assigned to every responsibility
4. [Title], is a short sentence describing the goal of the System Responsibility, using an infinitive verb
5. [Description], is the full description detailing, at an initially abstract level, what the system should do in order to fulfill this System Responsibility

Every System Responsibility list is accompanied by a Usability Elicitation Clusters / System Responsibilities mapper. This mapper helps analysts determine which of the proposed System Responsibilities for a feature are applicable for their project, depending on the results of the elicitation process.

Table 4.3-5 shows the structure of this Usability Elicitation Clusters / System Responsibilities mapper.

Table 4.3-5 Structure of the Usability Elicitation Clusters / System Responsibilities Mapper

Usability Elicitation Cluster	Dependent System Responsibilities
List of Usability Elicitation Clusters expressed as: [FUF ID]_EC_i	List of dependent system Responsibilities for each Usability Elicitation Cluster expressed as: [FUF ID]_SR_n [Name of System Responsibility n] ... [FUF ID]_SR_m [Name of System Responsibility m]

The UEC / SR Mapper is a two-column table. In the first column there is a list of all the elicitation clusters, identified their id and name. The second column contains all the System Responsibility dependencies for each clusters.

Every cluster may have one ore more System Responsibilities that depend on it, meaning that they will only be implemented *if* said cluster is present after elicitation.

The System Responsibilities are expressed in the following notation: [FUF ID]\_SR\_n [Name of System Responsibility n].

#### 4.3.1.3.2 Identification of system responsibilities for usability: Task description

In order to determine which System Responsibilities are to be included in the system, the development team (most commonly, the analyst role) must look at which Usability Elicitation Clusters were found to be applicable in the elicitation and analysis phase (see section 4.3.1.1.1.2). With the use of the UEC/SR mapper, the team then determines, for each applicable cluster, which System Responsibility is relevant to the system. After doing this for all applicable clusters, the team will have determined the subset of System Responsibilities that must be designed in this phase of development (i.e. solely those that embody what was elicited for each functional usability feature).

Appendix 9.1 shows an example of how this task is applied.

At this point the proposed process can continue on to the next task or stop, keeping the all the analysis and elicitation documents produced so far, namely the requirements document and and the use case model including usability

### 4.3.2 OO Software design activities for usability

This section covers the last three tasks of the proposed process, specifically those that make up the “OO software design activities for usability” sub-process, first shown in Figure 4.3-1, as part of the full Software Design process. Figure 4.3-4 below shows the tasks involved in this sub-process, now also depicting the usability guideline artifacts involved (dark gray).

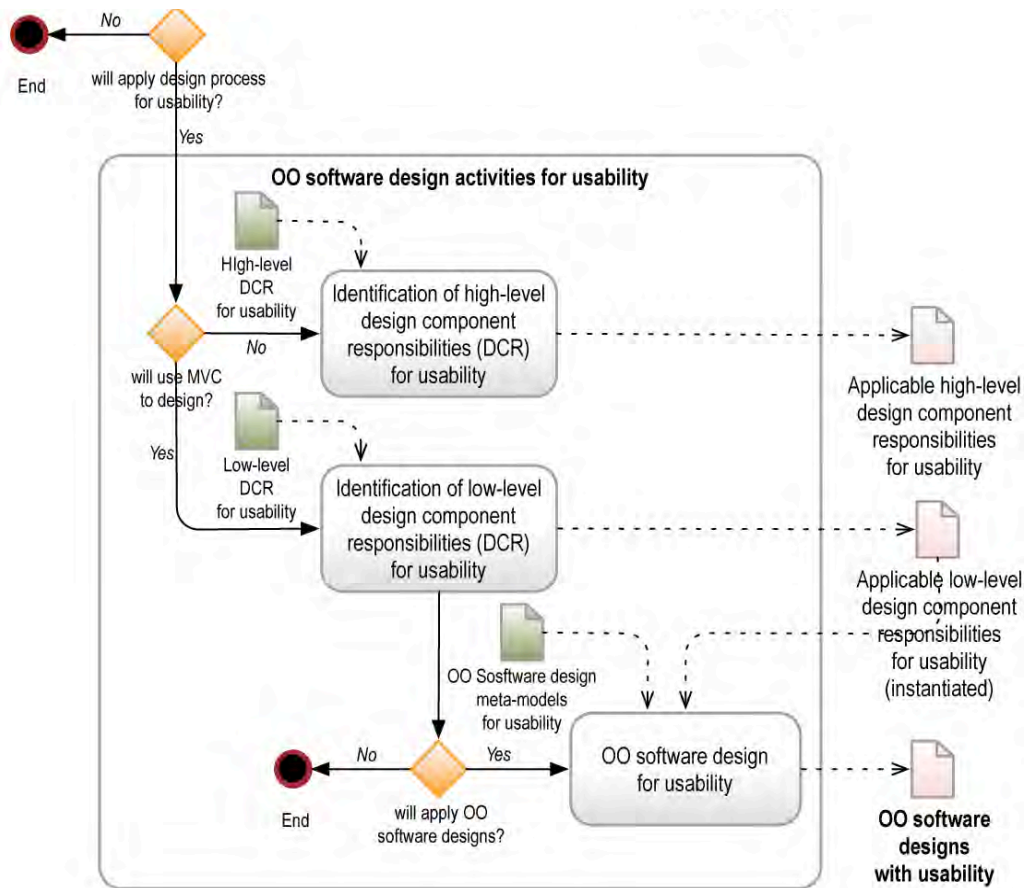


Figure 4.3-4 Object oriented designs for usability process with the use of the software usability guideline artifacts

#### 4.3.2.1 Identification of high-level design component responsibilities for usability

Based on the System Responsibilities that were found to be applicable in the previous task (see section 4.3.1.3), developers identify the corresponding high-level component responsibilities that are relevant to their systems. These express the more abstract System Responsibilities as groups of more finely-grained responsibilities, assigned to one or more parts (components) of the system to be developed.

This is an optional task, and is applicable for developments which do not use an MVC-based architecture and therefore would not benefit from the outputs of the next two tasks in this sub-process which are MVC-based (see sections 4.3.2.2.2 and 4.3.2.3). For developments which do use an MVC-based architecture, this task can be skipped and the process continues on the following task, described in section 4.3.2.2.2.

##### 4.3.2.1.1 Artifacts involved

The artifact involved in this task is the high-level design component responsibilities for usability, described below.

##### 4.3.2.1.1.1 High-level design component responsibilities for usability

The High-level software component responsibilities specify, at a lower abstraction level than that of System Responsibilities, what each part of the software would need to accomplish in order to fulfill the greater goal of a System Responsibility. A concrete example of this artifact is shown in section 5.2.2.1 of Chapter 5 for the Undo feature.

The High-level software component responsibilities table (see Table 4.3-6) is a two column table. In its first column, it lists all System Responsibilities, followed, in the second column, by textual descriptions of which high-level design components are needed to carry it out.

Furthermore, it specifies which responsibilities should be assigned to each component in order to do so.

Taking any broad system responsibility such as the ones presented in this work, and assigning tasks to multiple design components to carry it out is not a clear-cut task. Many different valid solutions can be proposed, as such a process inevitably involves a certain level of subjectivity.

Many questions arise when determining which components to consider, and, once even that step has been covered, the question of which of the defined components is better suited to take on which task to produce an adequate distribution of responsibilities (and future software design) arises.

There is no definite ‘optimal’ in this regard, but there is a long history of sound principles and patterns that, when applied correctly within a particular context, will guarantee to some extent certain desirable quality aspects when assigning responsibilities [37].

Due to the nature of this work, where each guideline is conceived to be used repeatedly and across multiple projects of diverse nature, the principles that were chosen have a common goal of preserving a high level of maintainability, clarity and reuse of the resulting components. These range from Gamma’s broader ‘low cohesion’ principle [37] to the use of more specific GoF patterns where appropriate, producing as a result an adequate distribution of responsibilities. It is one solution among many possible ones that could be created, but a solution with a certain guaranteed level of quality in the more relevant aspects to the successful use and applicability of the guidelines.

Table 4.3-6 shows the basic structure of the High-level software component responsibilities for usability.

Table 4.3-6 Structure of the High-level software component responsibilities for usability

System Responsibility	High-level design component responsibilities for usability
List of all System Responsibilities, expressed as: [FUF ID]_SR_i [Name of System Responsibility i]	[FUF ID]_ HLCR _n: [Textual description of High-level software component responsibilities. Suggested design components are italicized.] ... [FUF ID]_ HLCR _m: [Textual description of High-level software component responsibilities. Suggested design components are italicized.]

The first column lists all System Responsibilities, one per row, identified as [FUF ID]\_SR\_n [Name of System Responsibility i], where FUF ID is the identifier of the usability feature, SR stands for System Responsibility and i for the number of the responsibility, this is followed by the name of that responsibility.

The second column contains one or more textual descriptions per System Responsibility. These descriptions explain, in free text format, which generic components should be involved in fulfilling the current System Responsibility, and, in general terms, which tasks should be under the care of each one of those components. All component names are capitalized and italicized for clarity. Component responsibilities are identified as [FUF ID]\_ HLCR \_n: [Textual description] where FUF ID identifies the usability feature, HLCR stands for “high-level component responsibility” and n is the number of that responsibility.

#### 4.3.2.1.2 Identification of High-level design component responsibilities for usability: task description

In order to identify which of the High-level software component responsibilities for usability are applicable for their system, the development team must refer to the subset of System Responsibilities that was determined to be applicable in the previous task (see section

4.3.1.3). With this information, the team must look at the High-level software component responsibilities for usability table and select all of those that correspond to the applicable System Responsibilities.

The output of this task will be a subset of High-level software component responsibilities for usability. This output is however optional, and will be produced as an actual document only in the instances where the development team does not wish to continue applying the proposed process. This can be due to the fact that, for example, the development team for a particular project uses a vastly different architecture from MVC, and would thus find limited use for the lower-level usability artifacts.

Regardless of producing it as an output document however, the High-level software component responsibilities for usability serve as a blueprint for what the Low-level software component responsibilities will be, as is described in section 4.3.2.2.1 below.

Appendix 9.1 shows an example of how this task is applied.

#### 4.3.2.2 Identification of low-level design component responsibilities for usability

Based on the System Responsibilities that were found to be applicable (see section 4.3.1.3), developers identify the corresponding low-level component responsibilities that are relevant to their systems. These express the more abstract System Responsibilities as groups of more finely-grained responsibilities, assigned to individual classes and objects of the final design of the system to be developed.

##### 4.3.2.2.1 Artifacts involved

The artifact involved in this task is the Low-level design component responsibilities for usability, described below.

###### 4.3.2.2.1.1 Low-level design component responsibilities for usability

These low-level responsibilities are the least abstract representation of the System Responsibilities for each of the Functional Usability Features, as they describe how they would be designed (in terms of classes and methods) into a MVC-based system. A concrete example of this artifact is shown in section 5.2.2.2 of Chapter 5 shows for the Undo feature.

By examining each System Responsibility and further describing them as MVC elements, certain compromises must be made, the main one being the obvious loss of generality of the results, as a specific architecture needed be chosen in order to descend to the level of abstraction of objects, classes, instances, etc.

In choosing a specific architecture, we restrict the wide applicability of the resulting models, as only systems using said architecture would benefit from them. However, in order to bring results to the desired level of abstraction—one that development teams can use directly to implement their software, closer to software *code* than has ever been provided in previous works—the benefits of such a restriction outweigh their drawbacks.

In any case, the proposed process as a whole does not lose its generality and applicability, as it need not be used in its entirety to produce applicable results. As explained earlier, a development team not using MVC could still fully benefit from the proposed process all the way to the identification of High-level software component responsibilities for usability. Such a team would be able to assign these responsibilities to the appropriate components of their systems, instantiating them for whatever objects make up their specific architecture. This still represents a significant improvement over not having a process to go by when attempting to include the Functional Usability Features into their designs.



Teams who do use MVC (or equivalent), will fully benefit from this task as well as the next one, the OO Software design for usability, described in section 4.3.2.3.

Table 4.3-7 shows the structure of the Low-level software component responsibilities table.

Table 4.3-7 Structure of the Low-level software component responsibilities (for MVC)

System Responsibility	Objects			Figure
	Object 1	...	Object n	
List of all System Responsibilities, expressed as: [FUF ID]_SR_i [Name of System Responsibility i]	Description of each of the low-level responsibilities referencing their source high-level responsibility.			Reference to the Software Design Diagram Meta-models involved in this System Responsibility

Similarly to the High-level software component responsibilities table, the Low-level software component responsibilities table starts off with a list of System Responsibilities in its left column. These System Responsibilities are expressed in the same syntax as in the previous table: [FUF ID]\_SR\_i [Name of System Responsibility i].

The second column of this table is the Objects column. This column is divided in as many sub-columns as objects are involved in carrying out all System Responsibilities, (even though not *all* objects may be involved in *all* System Responsibilities). For each System Responsibility (row), the Object columns list the specific task(s) to be carried out by that Object (the cells of Objects not involved at all in the current responsibility are grayed out). Each task is preceded by a number, indicating the order in which the task must take place, across all objects.

Most of the objects listed in the Low-level software component responsibilities table represent actual objects to be implemented directly into a software system requiring the System Responsibility in question. There are, nevertheless, instances when responsibilities are assigned to an object of the system's own domain, which cannot exist nor be instantiated until the proposed process is applied to a specific project. Such objects are represented as *DomainObjects*), and emphasized here in italics for distinction from the rest of the concrete objects. Each guideline describes exactly how and when to substitute the actual domain object for this 'template'.

Finally, the last column in the Figure column, which simply provides a reference to the OO Software Design Meta-models that depict these low-level software component responsibilities for usability.

#### 4.3.2.2.2 Identification of low-level software component responsibilities for usability: Task description

In order to identify which of the Low-level software component responsibilities for usability are applicable for their system, the development team must refer to the subset of System Responsibilities that were determined to be applicable (see section 4.3.1.3). With this information, the team must look at the Low-level software component responsibilities for usability table and first select all of those that correspond to the applicable System Responsibilities.

As in the high-level case, the output of this task will be a subset of Low-level software component responsibilities for usability. This output is however optional, and will be produced as an actual document only in the instances where the development team does not wish to continue applying the proposed process. This can be helpful in instances when, for example, the development team is more open to receiving aid in textual form than in that of UML diagrams [4].

Appendix 9.1 shows an example of how this task is applied.

### 4.3.2.3 Object Oriented software design for usability

This is the final task of the proposed process and it entails designing the software application, or the parts of it being covered in the current iteration. During this design process, the development team applies the proposed software design meta-models for usability to include the required usability features

Section 4.3.2.3.1 below describes the guideline artifacts that are involved, followed by a step-by-step description of the task in section 4.3.2.3.2.

#### 4.3.2.3.1 Artifacts involved

The artifact used in this process is the software design meta-models for usability, described below.

##### 4.3.2.3.1.1 Software design meta-models for usability

The software design meta-models for usability are the graphic representation of the Low-level software component responsibilities for usability. They are made up of class and sequence diagrams written in UML that describe the way in which the functional usability feature they address is to be designed at the lowest level of abstraction possible, and ultimately implemented. A concrete example of this artifact is shown in section 5.2.2.3 of Chapter 5 shows for the Undo feature.

For every object listed in the Low-level software component responsibilities table, there will be one class in the Class and Sequence diagrams of this guideline. For every task listed in said table, the corresponding method(s) will be created within that class, and represented in both diagrams accordingly.

It is important to note that the proposed Usability Software Design Meta-models (as well as the chosen responsibility allocation) is but one of perhaps many possible solutions that could exist for each of the Functional Usability Features addressed. They represent, however, a sound solution which follows widely accepted design principles, and is as such an important contribution in this regard.

Figure 4.3-5 and Figure 4.3-6 show the structure of the Software Design Meta-models, in the form of Class and Sequence diagrams. Each Usability Software Design Guideline will have one single Class Diagram and one or more Sequence Diagrams, covering all of the System Responsibilities. However, the relationship between System Responsibilities and Sequence Diagrams is not one-to-one, as a group of System Responsibilities can be represented in a single Sequence Diagram and vice-versa.

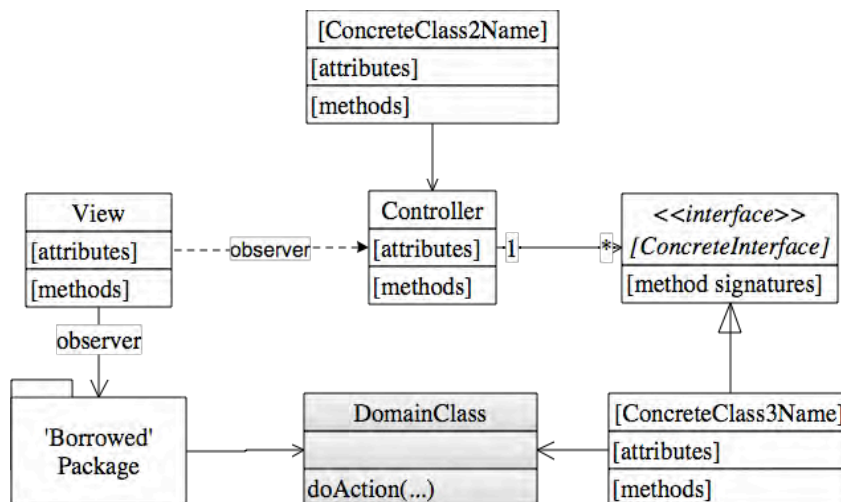


Figure 4.3-5 Structure of Software Design Meta-model. Class Diagram

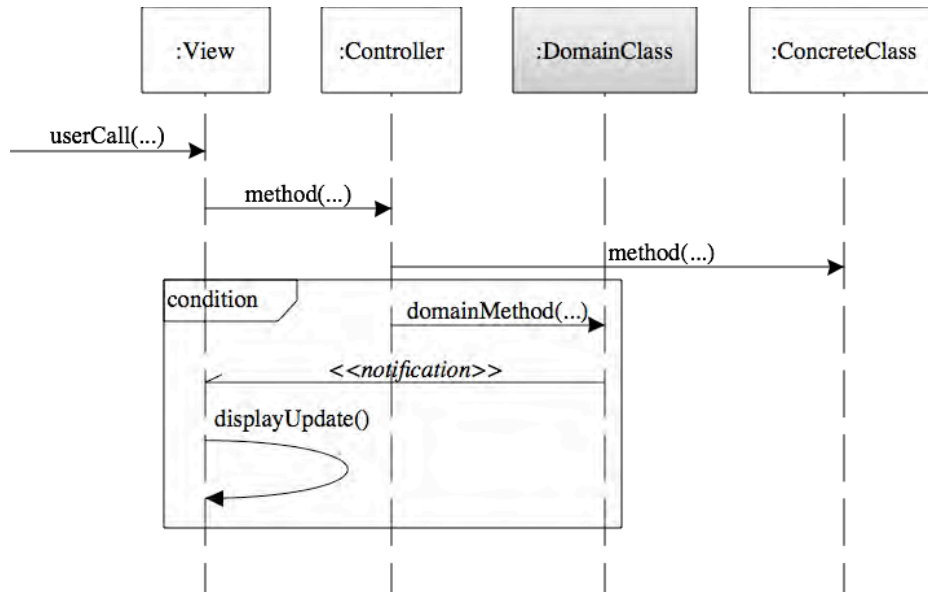


Figure 4.3-6 Structure of Software Design Meta-model. Sequence Diagram

Both diagrams follow the UML2 notation [21] with the following additions:

1. The 'template' DomainClass objects mentioned above are depicted in a different color (gray), to highlight them.
2. All classes belonging solely to the present feature are depicted in the feature's assigned color (see Appendix 9.4 for color legend).
3. Classes belonging to other features are depicted in the color of that feature to differentiate them. They are often represented as packages for clarity, and assumed to contain all or some of its classes.
4. Classes or methods belonging completely to an existing software design pattern are depicted in yet a different color (i.e. red for the Observer Pattern.)

#### 4.3.2.3.2 Object Oriented software design for usability: Task description

The Software Design Meta-models are to be used directly within the software designs when using a MVC<sup>6</sup> architecture. During the system design phase(s) of a project, class and sequence diagrams, likely among others, will be produced. For those not involving the Functional Usability Features, software designers will proceed as usual; creating their software designs as they normally would. However, when they must design the functionalities that pertain to any of the Functional Usability Features, they are expected to apply these Software Design Meta-models.

Most classes and methods present in the Software Design Meta-models can be directly copied into the system's own design to cover the Functional Usability Feature in question. However, there are 'template' classes in these meta-models that must be substituted by the system's own.

The output of this task, and the final output of the proposed process when applied in full, is a set of software design diagrams that includes all the elicited Functional Usability Features.

Appendix 9.1 shows an example of how this task is applied.

---

<sup>6</sup> Projects using other architectures such as PAC (where a 'translation' between MVC components and those of the other architecture is possible and common) the Low-level software design component responsibilities can still be used. Only this additional step of 'translation' will be required before moving on to the design itself.

## 4.4 Process Automation

Our proposed process can be applied manually or with the aid of our proposed Support Tool for Usability-Oriented Development. When applying it manually, the parties involved in use printouts (or digital versions) of the proposed guidelines, and in the automated case the process is applied with the aid of the tool.

The purpose of this proposed tool is to help development teams apply the proposed process more easily and efficiently. In the case of analysts, they would not need to rely on the paper versions of the guidelines to conduct the stakeholder discussions, and the results of these discussions can be directly input into the tool (to later transfer them to whatever format of requirements document is used within the project. In the case of software designers the benefit is even higher in terms of efficiency. With the manual process, UML diagrams that need to be copied and instantiated manually one by one from paper into a UML design tool. When using the proposed tool, however, designers receive a digital version (XML file) of each diagram, already including only the elements that are relevant to their project. Designers would only need to open these files with one of many UML editing tools with support for XML, and include them directly into their projects.

The proposed tool is accompanied by a prototype pre-load application, developed to be used only once prior to the use of the tool. Its purpose is transforming the paper guidelines into sets of XML files that are readable by the tool, as explained later in this chapter. Once the guidelines have been converted, they are loaded into the tool, allowing software developers to interact with them during the proposed process. Both the pre-load application and the tool are described in the next two sections.

### 4.4.1 Pre-load application

The purpose of this application is to transform a Usability Guideline for Software Development into digital files to be fed into the proposed tool. This task is performed only once, before first using the proposed tool. This pre-loading application also allows for the inclusion of new guidelines should any be developed in the future.

Upon opening the pre-load application, the user is shown with a screen as the one shown in Figure 4.4-1, for loading each of the discussion items of the UEG.

The screenshot shows a graphical user interface for a pre-load application. At the top, there are four text input fields labeled: 'Valor de la pregunta:', 'Numero de pregunta:', 'NodoSi:', and 'NodoNo:'. Below these fields are two columns of controls. The left column has a button labeled 'Añadir AccionSi' above a list area labeled 'AccionesSi:'. Below the list area is a button labeled 'mostrar acciones Si'. The right column has a button labeled 'Añadir AccionNo' above a list area labeled 'AccionesNo:'. Below this list area is a button labeled 'mostrar acciones No'. At the bottom center of the window is a button labeled 'Aceptar'.

Figure 4.4-1 Loading the Usability Elicitation Guideline into the pre-load application

The first four input fields in this screen are for loading the text of the discussion item, its identifier and the identifier(s) of the discussion item(s) that follow the present item. The two buttons that follow are for adding these subsequent items (labeled “si” and “no” for “yes” and “no” in the case of binary questions that branch in two directions, depending on whether they are answered in the positive or negative). Once the discussion items are added, they show up in the text boxes below the buttons. Advanced users may click on the buttons below the text boxes, labeled “mostrar acciones Si/No” to view the underlying XML code that is generated for each added discussion item.

Once this information has been filled out, the user clicks on the Accept button at the bottom of the screen and the present discussion item is closed and added to the main XML file.

When the discussion item being added can be responsible for eliminating an entire cluster of items, the user must associate it with the corresponding System Responsibility/ies for Usability that would be discarded. Figure 4.4-2 shows the screen where this association is performed. In this screen the user must enter the discussion item number, the name of the table where the system responsibilities are present (which must match the name of the source .xls file where the system responsibilities are stored, i.e. “Undo.xls”) and the row from that table to eliminate. Finally, clicking the lower-most button on the screen stores this information.

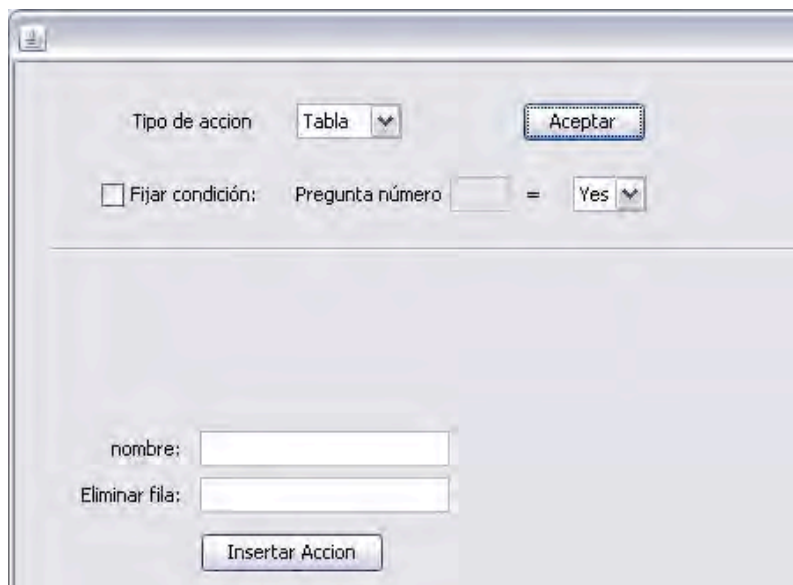


Figure 4.4-2 Pre-load Application: Associating elicitation clusters and system responsibilities

The user must repeat this task for every discussion item in the Usability Elicitation Guideline.

Once all the discussion items have been loaded, the user must proceed to load the UML models. First is the Use Case meta-model. For this, the user selects the option to add an use case (“Añadir caso de uso”) and clicks the Accept button, as shown in Figure 4.4-3. This will present him with a set of text fields where he is to introduce one use case at a time, along with its associations with other elements of the diagram.

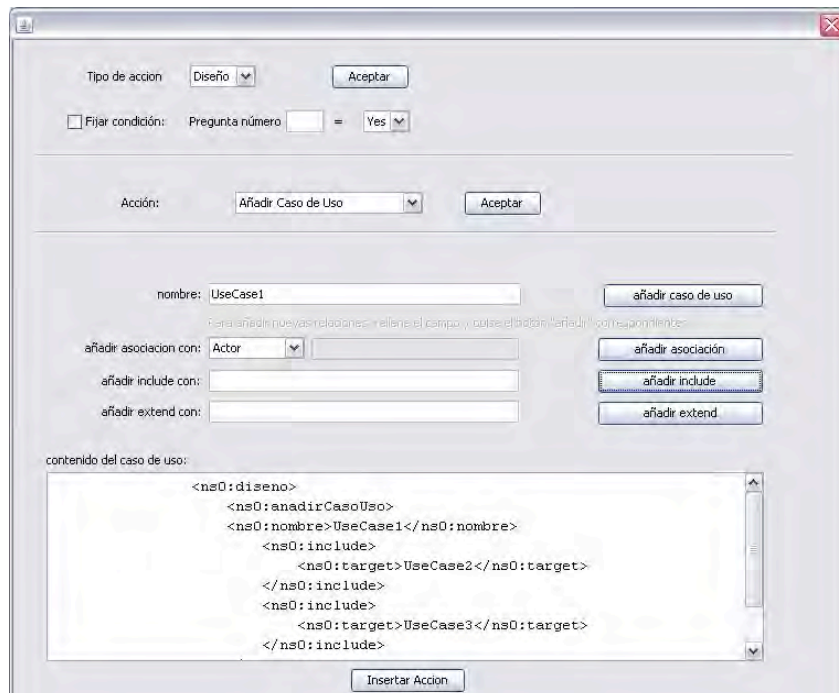


Figure 4.4-3 Pre-load tool. Loading use cases.

In the first text field the user would input the name of the use case and would then click on the button to add the use case to the XML file. The contents of the XML file is shown in the text box at the bottom of the screen. Any associations would be added by selecting from the drop down box the type of element that the use case is associate with (i.e. actor, another use case, etc.), followed by the option to label the association as being of the type “include” or “extend”. Once all this information is filled in for the present use case, the user clicks on the button at the bottom of the screen to store it in the XML file. Users would perform this action for all use cases in the meta-model of the Usability Diagram that they are pre-loading.

Similarly to Use Cases, elements of the software class diagram are added through the screen shown in Figure 4.4-4.

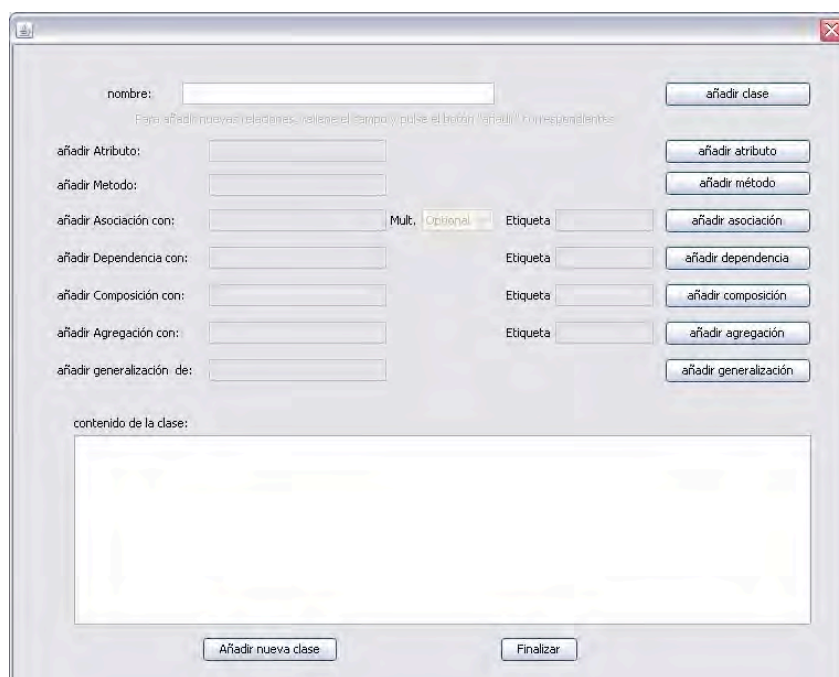


Figure 4.4-4 Pre-load tool. Loading class diagram

The user must fill out a set of text fields for the name of the class and any associations it may have with other diagram elements. Clicking in the bottom-left button of the screen “Añadir nueva clase” adds the current class to the XML file (as shown on the text box above the buttons) and clears the text fields for adding a new one. Once all classes are added, the user clicks on the bottom-right button of the screen “Finalizar” to end.

Finally, the sequence diagrams are added through the screen shown in Figure 4.4-5.

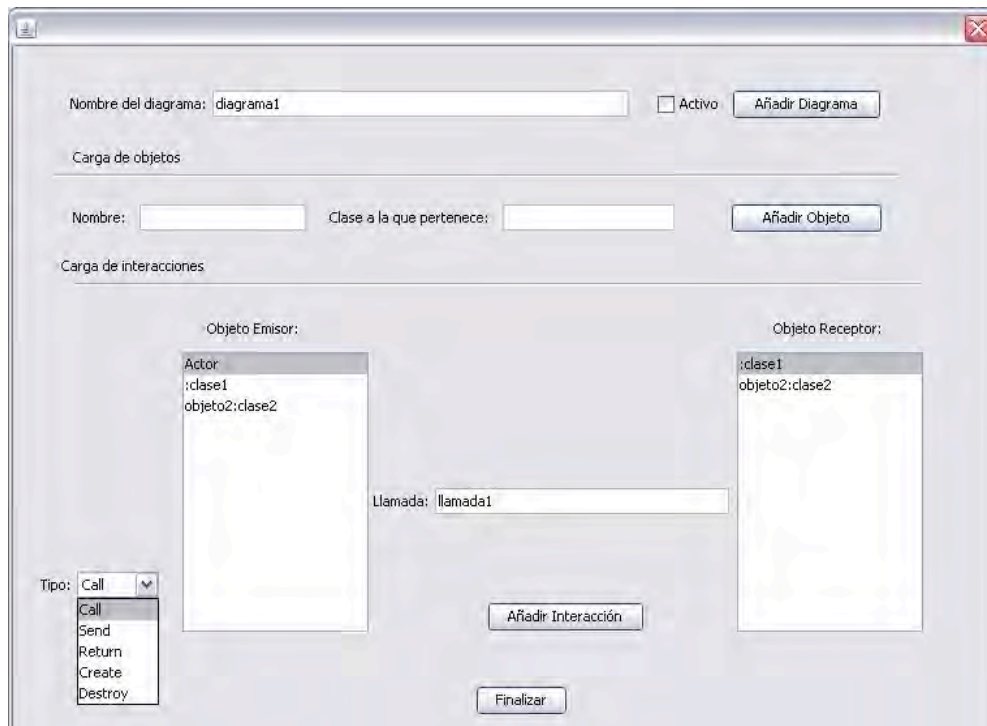


Figure 4.4-5 Pre-load tool. Loading sequence diagrams

For every sequence diagram, the user inputs the name of the diagram in the first text field. In the next two fields, the user inputs all classes and instances that will be present in the sequence diagram. Once those are added, the user selects them one by one from the list generated on the left and associates it (by a method call, return message, create directive etc) with the necessary ones on the right-hand list, until all the associations are complete.

Once the last diagram has been added, the user selects the option to generate the output files for the present Usability Guideline. Once done for all the Usability Guidelines, the Tool is ready to be used as explained in the next section.

#### 4.4.2 Support Tool for Usability-Oriented Development

When using the software tool, the analyst logs into the application and selects the first functional usability feature to address (Figure 4.4-6) during the elicitation process. By hovering over the available features, the screen presents the user with a description of it, including the features intent, the problem it addresses and the contexts within it is applicable.



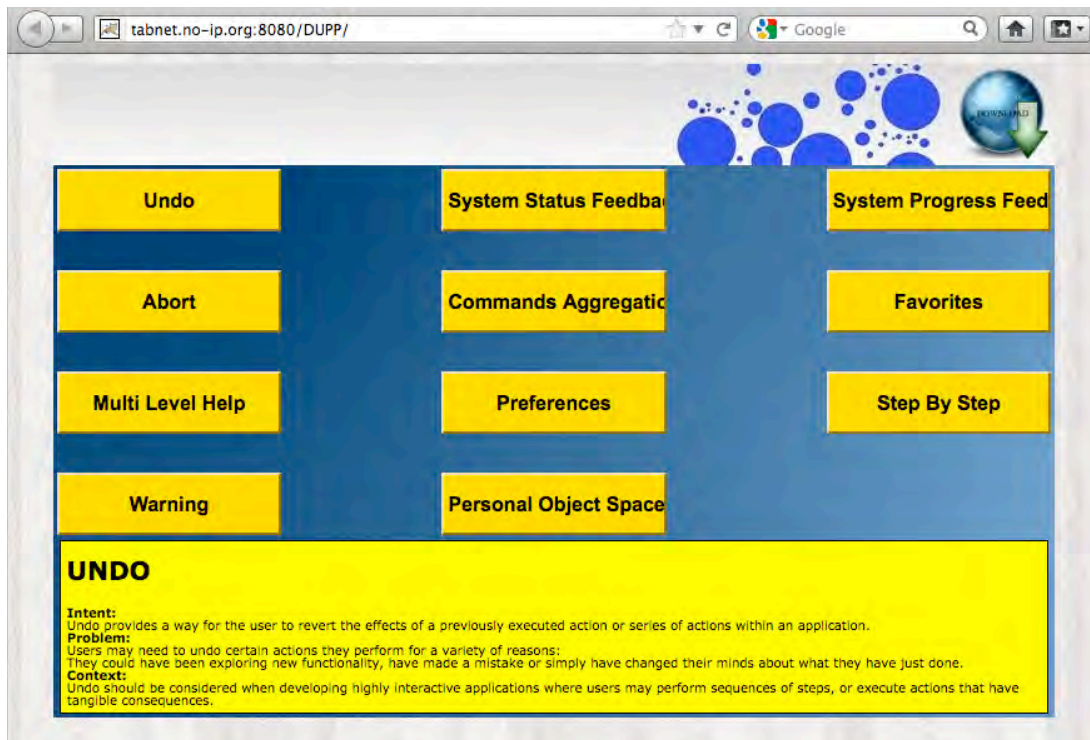


Figure 4.4-6 Software tool: Functional usability feature selection

Once the analyst has selected a feature, the application loads the guideline and presents a view of the Elicitation Clusters for that feature, along with text fields for recording the output of each individual discussion item (Figure 4.4-7). In this example, the selected Functional Usability Feature will be Undo.

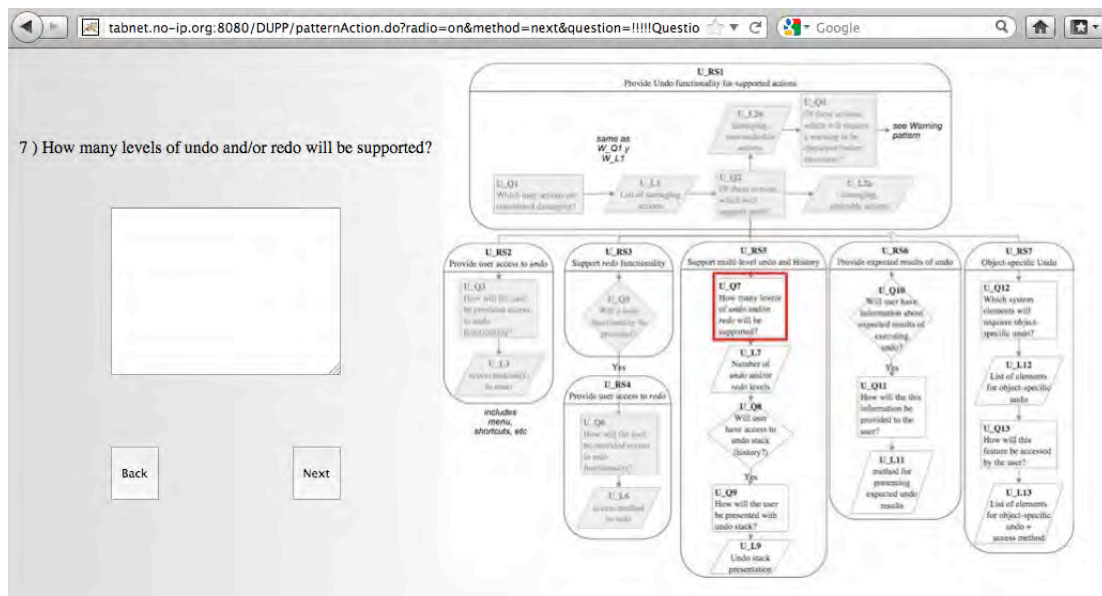


Figure 4.4-7 Software tool: Elicitation clusters for Undo feature

As the discussions move forward, the active discussion is highlighted in red, and discussion items are marked as grayed out as they are 'covered', highlighting the next one in the sequence. When the result of a discussion item discards one or more future items they are grayed out and skipped automatically by the tool. In this example, the stakeholders have stated the need for an undo feature of three levels, global and over specific objects, without redo, with no viewable history list and no "smart menus" (see Undo guideline in Chapter[G] for a definition of these terms)



If during the discussion process either the analyst or any of the stakeholders need more information on a given discussion item, clicking on the (?) icon will present information pertaining to that item from the Elicitation Guideline (i.e. underlying HCI recommendation, usage examples, etc.)

Once all the discussion items are covered, and together with the pre-loaded information for each guideline, the tool now has the necessary information needed to generate all the outputs. The user is then returned to the first screen (Figure 4.4-6) and clicking on the “Download” button will produce a zip file containing all the output files as explained below.

The first downloadable file is a plain-text file containing the answers given during the elicitation process. For the present example, the resulting file is shown in Table 4.4-1.

Undo - Elicitation Results
List of damaging actions Deleting files
Damaging undoable actions Deleting files
Access method(s) to undo Undo will be accessed through the Edit menu
Number of undo and/or redo levels Three
List of elements for object-specific undo All graphic elements
List of elements for object-specific undo + access method right-click over element

Table 4.4-1 Software tool output: Elicited information regarding usability

For the next downloadable product, the tool first accesses the pre-loaded relationships between applicable Usability Elicitation Clusters and System Responsibilities for Usability. With this information, it generates a file containing only the applicable System Responsibilities, making it available for download as an editable spreadsheet document, should the user be interested in this output. The actual file produced for this example is shown in Figure 4.4-8, where only the first, second and seventh System Responsibilities for Usability are applicable.

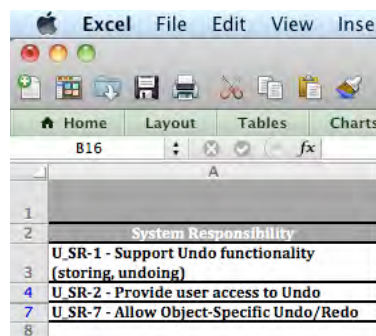


Figure 4.4-8 Software tool output: System Responsibilities

With the information about which System Responsibilities are applicable, the application automatically generates documents containing the applicable high and low level software component responsibilities for usability, making them optionally available for download as editable spreadsheet documents as well. These two files, as produced for this example, are shown in Figure 9.1-2 and Figure 9.1-3, respectively.

System Responsibility	Generic Component Responsibilities
U_SR-1 - Support Undo functionality (storing, undoing)	<ul style="list-style-type: none"> <li>- The component responsible for handling user events (UI) must listen for calls to actions and order their execution</li> <li>- The component in charge of delegating actions (if any) should determine whether the action is undoable or not, from a pre-established list.</li> <li>- If the action to execute is undoable, it must first be encapsulated as an instance of a Command Component, together with any pertinent state information and the necessary actions needed to revert its effects.</li> <li>- Such an instance is then stored in a History Component, responsible for keeping a single (ordered) collection of all executed undoable actions.</li> <li>- After encapsulation, the Domain Component is then free to execute the invoked action</li> <li>- The UI Component must listen for calls to the Undo action (if available) and order its execution</li> <li>- The History Component must then retrieve the last*/current** Command, without discarding it, and order it to 'undo' itself.</li> <li>- The Command, in turn, executes the necessary actions, using the stored state information, to return the system to the state preceding its execution</li> </ul>
U_SR-2 - Provide user access to Undo	<ul style="list-style-type: none"> <li>- The UI Component is responsible for providing the mean(s) through which a user can invoke the undo feature. The Undo action should only be available when at least one undoable action has been executed during application up-time</li> </ul>
U_SR-7 - Allow Object-Specific Undo/Redo	<ul style="list-style-type: none"> <li>- The UI Component must listen for calls to the Undo/Redo action over a particular object (if available) and order its execution.</li> <li>- The History Component must then retrieve the last*/current** Command executed over that object, without discarding it, and order it to Undo/Redo itself.</li> <li>- The Command, in turn, executes the necessary actions, using the stored state information, to return the object to the state preceding/following its execution.</li> </ul>

Figure 4.4-9 Software tool output: High-level software component responsibilities

System Responsibility	View	Controller	ConcreteCommand	HistoryList	DomainClass
U_SR-10000 Provide Undo functionality (storing/undoing)	<ol style="list-style-type: none"> <li>1. The View must listen for invocation of actions. Upon reception, it must notify the Controller of said action</li> </ol>	<ol style="list-style-type: none"> <li>2. The Controller must determine if the invoked action is undoable. In such case it must call the execute() method of the corresponding ConcreteCommand object (otherwise invocation goes directly to the DomainClass).</li> <li>3. The Controller must then clone() said ConcreteCommand and add() it to the HistoryList.</li> </ol>	<ol style="list-style-type: none"> <li>4a. Upon call to its execute() method, the ConcreteCommand first stores the necessary state information in its local variables. It then calls the appropriate method in the corresponding DomainClass (what was originally invoked)</li> </ol>	<ol style="list-style-type: none"> <li>4b. The HistoryList saves the cloned ConcreteCommand atop its collection (so it can later be available to undo)</li> </ol>	<ol style="list-style-type: none"> <li>5a. The DomainClass executes the appropriate methods carry out what was originally invoked the user through the View.</li> </ol>
U_SR-20000 Provide user access to Undo	<ol style="list-style-type: none"> <li>1. The View must listen for invocation of the Undo action. Upon reception, it must notify the Controller.</li> </ol>	<ol style="list-style-type: none"> <li>2. The Controller orders the HistoryList to undo the last action</li> </ol>	<ol style="list-style-type: none"> <li>4. Upon call to its undo() method, the ConcreteCommand calls the necessary methods in DomainClass (with any needed state information, stored upon execution) to revert its effects.</li> </ol>	<ol style="list-style-type: none"> <li>3. The HistoryList determines the ConcreteCommand to undo and calls its undo() method.</li> </ol>	<ol style="list-style-type: none"> <li>5. The DomainClass executes the methods invoked by ConcreteCommand</li> </ol>
U_SR-70000 Allow Object-Specific Undo/Redo	<ol style="list-style-type: none"> <li>1. The View must listen for invocation of the Undo/Redo action over a specific object. Upon reception, it must notify the Controller.</li> </ol>	<ol style="list-style-type: none"> <li>2. The Controller orders the HistoryList to undo/redo the last action invoked over said object.</li> </ol>	<ol style="list-style-type: none"> <li>4. Upon call to its undo()/redo() method, the ConcreteCommand calls the necessary methods in DomainClass (with any needed state information) to revert/reinstate its effects.</li> </ol>	<ol style="list-style-type: none"> <li>3. The HistoryList determines the ConcreteCommand to undo/redo by scanning the collection for the latest/current one invoked over the object and calls its undo() method.</li> </ol>	<ol style="list-style-type: none"> <li>5. The DomainClass executes the methods invoked by ConcreteCommand</li> </ol>

Figure 4.4-10 Software tool output: Low-level software component responsibilities

Using the pre-loaded information and the results of the elicitation process, the tool also produces an instantiated version of the Usability use cases for usability meta-model, containing only the applicable use cases for what was elicited. They are produced in standard XMI format for UML and are readable by any UML editing application with the option to import such files. Should the development team be interested in modeling use cases they can import this file into any freely available UML modeling application, such as StarUML, and edit it directly, as shown in Figure 4.4-11.

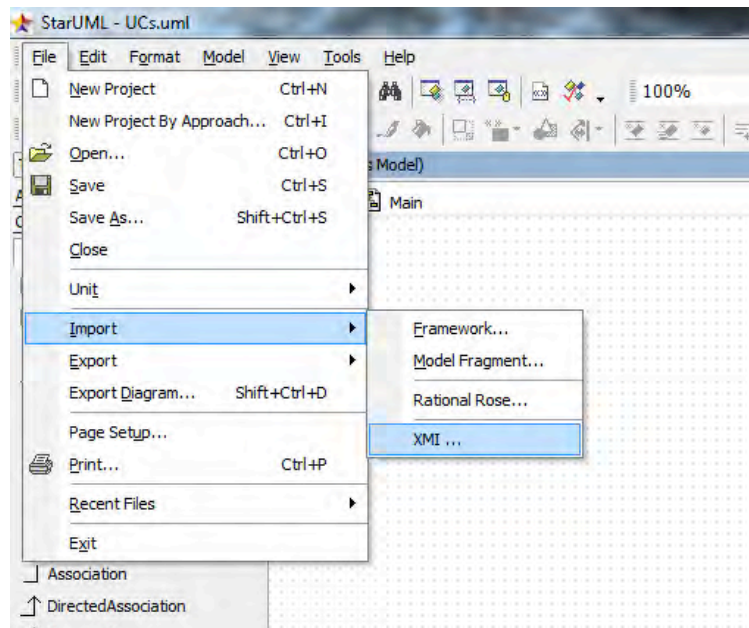


Figure 4.4-11 Software tool: Importing XML output for UML diagrams into StarUML

For this example, the resulting use case diagram, including only the relevant use cases, is shown in Figure 4.4-11<sup>7</sup>. When compared to the original Usability Use Case Meta-model in page 77, we can see that the use cases relating to redo and other features discarded in elicitation, have been excluded.

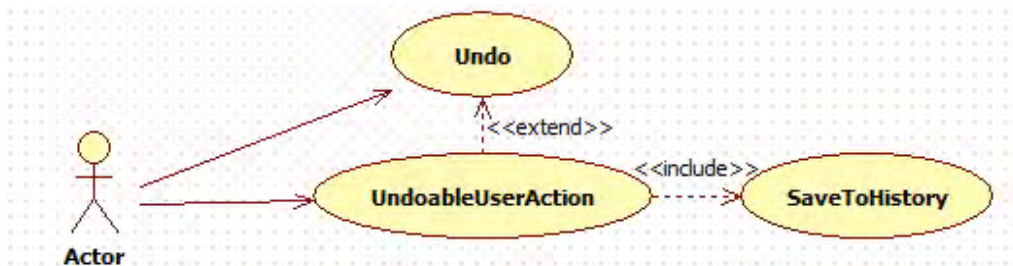


Figure 4.4-12 Software tool output: Usability use cases meta-model

Finally, and based on the low-level responsibilities that were determined to be applicable, the system generates all relevant design diagrams and makes them available for download as an editable UML file as well. The class diagram for this example is shown in Figure 4.4-12, where, when comparing to the original Software Design Meta-models for usability in page 86, we can see that the methods pertaining to redo aren't present, as well as the redo-related Exceptions.

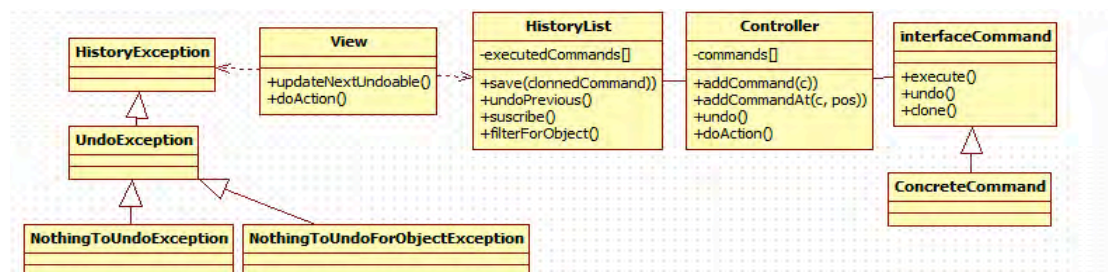


Figure 4.4-13 Software tool output: class diagram

<sup>7</sup> All generated XMI files, when opened by StarUML, show the graphic elements correctly on screen yet at random placements. For this example, the elements have been moved to resemble their original locations on the Usability Guidelines.

## CHAPTER 5. USABILITY GUIDELINES FOR SOFTWARE DEVELOPMENT

### 5.1 Introduction

This chapter contains the Usability Guidelines for Software Development proposed for the eleven Functional Usability Features addressed in this work.

The **Undo** guideline, which covers the user's need to revert the effects of a previously executed action, or series of actions, within an application, is described in section 5.2.

Section 5.3 presents the **Abort** guideline. Abort entails canceling on-going commands within an application and to exit the application altogether.

In section 5.4 the **Step-by-Step** guideline is presented, covering allowing tasks with multiple steps to be represented as a series of navigable windows

The **Progress** guideline is described in section 5.5. This usability feature aims at providing the user with accurate visual feedback on the progress of the current task.

The guideline for the **System Status Feedback** feature, which provides the user with information on the different statuses the system might be in at any given time, is detailed in section 5.6.

Section 5.7 presents the guideline for the **Warning** feature. This feature entails providing different alert types upon execution of sensitive actions.

The **Multi-level Help** guideline is described in section 5.8. Multi-level Help allows the user to access textual help features in different levels of detail throughout a software application

In section 5.9 the **Commands Aggregation** guideline is presented. This usability feature allows the user to aggregate commands into macro-like structures for ease of batch execution.

The **Preferences** guideline, covering the user's need to receive accurate visual feedback on the progress of the current task, is described in section Section 5.10.

The guideline for the **Favorites** feature, which allows the user to bookmark and keep a collection of favorite places within an application, is presented in section 5.11.

Finally, 5.12 presents the guideline for the **Personal Object Space** feature. Personal Object Space covers the users needs to arrange and manipulate objects graphically on screen.

Every one of the aforementioned sections, aside from presenting the Usability Guideline for Software Development that it covers, describes every artifact of that guideline in detail. As mentioned in Chapter 3, these artifacts cover the analysis and design phases. For the analysis phase, these artifacts are:

- The Usability Elicitation Guideline
- The Usability Elicitation Clusters
- The Usability Use Cases Meta-model
- The System Responsibilities

The artifacts intended for the design phase are the following:

- The High-level design component responsibilities
- The Low-level design component responsibilities for MVC
- The Software Design Meta-models

Each of these is described for every guideline in the following eleven sections. Furthermore each guideline has an associated color in which it's use cases and classes are depicted. As the usability features covered by the guidelines often overlap, color coding is needed to differentiate the elements of a specific feature from those that belong to other features. The colors assigned to each feature throughout this chapter are shown in

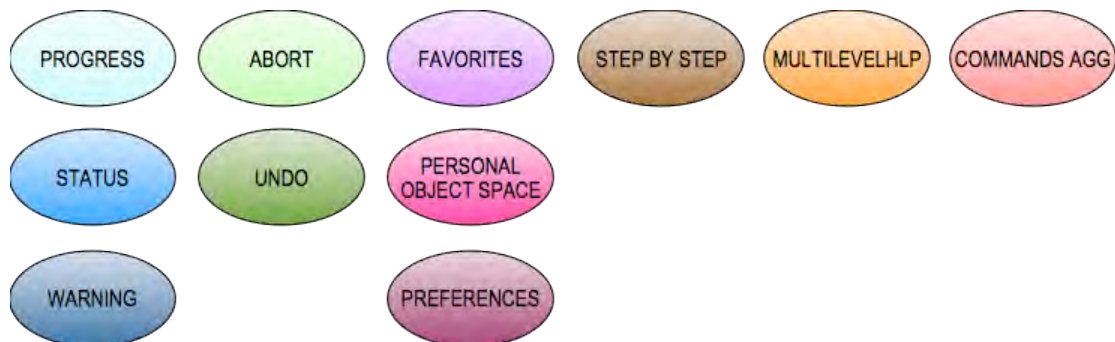


Figure 5.1-1 Color Legend for the functional usability features covered in this work

Finally, elements taken from existing software design patterns are also depicted in different colors, **red** for the Observer Pattern and **yellow** for the Command Pattern [37].



## 5.2 “Undo” Usability Guideline for Software Development

The Undo Functional Usability Feature covers the user’s need to revert the effects of a previously executed action, or series of actions, within an application. Undo allows users the liberty to explore the application’s functionality, to make mistakes or to change their minds during execution without being penalized for it (by being allowed to undo undesired results).

The Usability Guideline for Software Development is made up of Analysis and Design artifacts, described for the Undo feature in sections 5.2.1 and 5.2.2, respectively.

### 5.2.1 Usability Guideline for Software Development: Analysis artifacts

There are four artifacts to be used during the analysis phase: the Usability Elicitation Guideline, the Usability Elicitation Clusters, the Usability Use Case Meta-models, the System Responsibilities for Usability. These artifacts are described in the following four sections.

#### 5.2.1.1 Usability Elicitation Guideline

Table 5.2-1 shows the Usability Elicitation Guideline for the Undo Functional Usability Feature. In this guideline, there are six HCI recommendations, described below.

##### 5.2.1.1.1 Undoing actions

HCI authors suggest that the Undo functionality is necessary within applications where users can perform actions with potentially permanent consequences. This would encourage users to explore and learn the application functionalities without fear of making irrevocable mistakes (U\_HCI-1). What the terms ‘permanent consequences’ and ‘irrevocable mistakes’ mean exactly will depend greatly on the application being developed and therefore must be clearly defined by the project stakeholders (U\_ELAB-1) during elicitation. Only after doing so it will be necessary to discuss which system actions will indeed be required to be undoable (U\_Q-1 to U\_Q-3). In Table 5.2-1, example U\_EX-1: “Costly vs. undoable actions” describes an example for this HCI recommendation.

##### 5.2.1.1.2 Providing warnings for non-undoable actions

Some actions are simply not undoable, so the need for a warning when users attempt to perform them is discussed (U\_HCI-2). This type of warnings represent what is defined as an Authorization within the Warning guideline (U\_ELAB-2). If such actions exist within the application to be developed, stakeholders must pinpoint them (U\_Q-4) and determine how to address them, according to the Warning guideline. Example U\_EX-2: of Table 5.2-1, “File deletion w/warning”, describes an example for this HCI recommendation.

##### 5.2.1.1.3 Redoing actions

With the ability of undoing an action comes the possibility of redoing it (U\_HCI-3), which entails reverting the effects of an initial Undo invocation, if any (U\_ELAB-3). Similarly to the elicitation of the undo functionality, the discussions proposed for this HCI recommendation (U\_Q-5 and U\_Q-6) will determine if and how to provide the redo functionality. Example U\_EX-3: “Redoing in MS Word” describes an example for this HCI recommendation, in Table 5.2-1.

##### 5.2.1.1.4 Maintaining history log

Also, the need to undo more than just the latest action is explained, and the possibility of undoing a *series* of actions invoked by the user is suggested through the use of some kind of log or history (U\_HCI-4). This history may be linear or have a complex tree structure (U\_ELAB-4) and whether or not it’s required, its potential size and the ways to access it are discussed with stakeholders in discussions U\_Q-7 to U\_Q-9. In Table 5.2-1, example U\_EX-4: “MS Word’s undo Stack” describes an example for this HCI recommendation.

#### 5.2.1.1.5 Supporting 'smart-menus'

Smart menus that tell the user which action is next to undo (U\_HCI-5) are also part of the recommendations put forth by the HCI in this regard. They are easily provided within an application that holds a log or history (U\_ELAB-5) and stakeholders must decide if it's needed, and if so, how to present it to the user (U\_Q-10 and U\_Q-11). Example U\_EX-5: "Smart Menus in drawing app" describes an example for this HCI recommendation, in Table 5.2-1.

#### 5.2.1.1.6 Providing object-specific undo

Certain applications require that users be able to undo the latest action invoked *over a specific object* rather than over the application as a whole (U\_HCI-6). This functionality is fairly straightforward and can be achieved by filtering the log or history of executed actions (U\_ELAB-6). Discussion items U\_Q-11 to U\_Q-13 will determine which system objects (if any) will require this type of specific undo and how to provide the functionality to the user. Example U\_EX-6: "UML Design Program" in Table 5.2-1, describes an example for this HCI recommendation.

Table 5.2-1 Usability Requirements Elicitation Guideline. Undo.

Identification			
Name	Undo		
Family	Undo/Cancel		
Aliases	Multi-Level Undo [42]; Undo [49]; Global Undo / Object-Specific Undo [36]; Allow Undo [11]		
Intent			
Undo provides a way for the user to revert the effects of a previously executed action or series of actions within an application.			
Problem			
Users may need to undo certain actions they perform for a variety of reasons: They could have been exploring new functionality, have made a mistake or simply have changed their minds about what they have just done.			
Context			
Undo should be considered when developing highly interactive applications where users may perform sequences of steps, or execute actions that have tangible consequences.			
Interrelationships			
When including Undo in an application, in order to accommodate damaging actions that cannot be undone (and must thus trigger a warning to the user) the Warning feature must be considered			
HCI Recommendation	Elaboration	Discussions with Stakeholders	Usage Examples (optional)
<p>U_HCI-1 Undoing Actions</p> <p>Users typically explore functionality of an application but do not want to be "punished" when selecting unwanted functions [49]. The ability to undo a long sequence of operations lets users feel that the interface is safe to explore. While they learn the interface, they can experiment with it, confident that they aren't making irrevocable changes – even if they accidentally do something bad.</p> <p>So, first decide which operations need to be undoable [42]: Any action that might change a file (i.e. anything that could be permanent) should be undoable, while transient or view-related states often are not.</p> <p>In any case, make sure the undoable operations make sense to the user. They can be specific functions or a meaningful group of actions (for example, changing the printer settings) [49]. Be sure to define and name them in terms of how the user thinks about the operations, not how the computer thinks about them.</p>	<p>U_ELAB-1: Choosing undoable actions</p> <p>All actions with important consequences should provide the undo feature, save for those for which there are greater impeding factors present.</p> <p>For actions which are expensive to revert, The cost/benefit ratio of providing it with an undo feature should be evaluated.</p> <p>The term 'important consequences' must be clearly defined before deciding which actions will be undoable.</p>	<p>U_Q-1 Which user actions are considered to have important consequences?</p> <p>U_Q-2 Of these actions, which will support undo?</p> <p>U_Q-3 How will the user be provided access to the undo functionality?</p>	<p>U_EX-1: Costly vs. undoable actions</p> <p>Deleting a file from a system will most likely have important consequences and should be undoable</p> <p>Sending an email has important consequences (the email reaches the other party), but it is not directly undoable. Only such a limitation should keep a system from providing the undo feature.</p> <p>Deleting files from a hard drive, though undoable, tends to be an expensive operation to revert.</p>
<p>U_HCI-2 Warnings for non-undoable actions</p> <p>If a command has side effects that cannot be undone, warn the user before executing the command and do not queue it [49]</p>	<p>U_ELAB-2: Warnings: Authorization</p> <p>Most likely a warning of the type Authorization will be required. See warning pattern for the warning process.</p>	<p>U_Q-4 Of the damaging actions that cannot be undone, which will require a warning to be displayed to the user?</p>	<p>U_EX-2: File deletion w/warning</p> <p>If deleting a file from a hard drive is an undoable operation, the user will need to be warned (OK - Cancel style authorization) before the deletion is carried out.</p>



<p>U_HCI-3 Redoing Actions</p> <p>Users tend to explore a navigable artifact in a tree-like fashion, going down paths that look interesting, then back up out of them, then down another path [42]. So, an undo stack will need to be created. Each operation goes on the top of the stack as it is performed; each Undo reverses the operation at the top, then the next,... The undo concept must also include the concept of redo needed in case the user backs up too many steps [36]. Redo works its way back up the stack in a similar manner. The best undo should preserve the tree structure of the command execution sequence.</p>	<p>U_ELAB-3: Redo: Availability</p> <p>Redo should only revert the effects of the latest applied Undo. Some existing software currently use the Redo feature as a way to repeat the execution of any command. In the way in which we refer to it here, we strictly mean Redo as a way to revert a previously executed Undo. In this context, when no Undo command has been executed, Redo should not be available</p>	<p>U_Q-5 Will a redo functionality be provided?</p> <p>U_Q-6 How will the user be provided access to the redo functionality?</p>	<p>U_EX-3: Redoing in MS Word</p> <p>When undoing the changing of the font of a paragraph in MS Word, executing Redo will revert the text to its original font (before executing undo)</p>
<p>U_HCI-4 Maintaining History Log</p> <p>Often users want to reverse several actions instead of just the last action [49]. So, the stack should be at least 10 to 12 items long to be useful, and longer if you can manage it. Long-term observation or usability testing may tell you what your usable limit is (Constantine and Lockwood assert that more than a dozen items is usually unnecessary, since "users are seldom able to make effective use of more levels". Expert users of high-powered software might tell you differently. As always, know your users.</p> <p>Most desktop applications put Undo/Redo items on the Edit menu. Show the history of commands so that users know what they have done [49]. Undo is usually hooked up to Ctrl-Z or its equivalent</p>	<p>U_ELAB-4: History: Stack size and type</p> <p>Ideally undo/redo should use a tree structure instead of a stack structure to keep record of the actions, however the tree structure requires an important coding effort, so have this in mind when determining which kind of structure will be needed to keep record of the actions to be undone/redone. Notice that the system may have a global stack with a concrete size, or depending on the system, the size of the stack may be different for different functionalities.</p>	<p>U_Q-7 How many levels of undo and/or redo will be provided?</p> <p>U_Q-8 Will the user have access to the undo stack (history)?</p> <p>U_Q-9 If so, how will the user be presented with the undo stack?</p>	<p>U_EX-4: MS Word's undo Stack</p> <p>MS Word and others provide users with a visual list (stack) of the latest operations executed within the application. Within this stack, users can not only view the operations in the order in which they would be undone, but they can select an operation deep within the stack and undo it, along with every operation that was executed after it.</p>
<p>U_HCI-5 Supporting smart Menus</p> <p>The most well-behaved applications use Smart Menu Items to tell the user exactly which operation is next up on the undo stack.</p>	<p>U_ELAB-5: Smart Menus and History</p> <p>Smart menus are tightly related to the command history (stack). If one is kept, it's relatively simple to offer smart menus different functionalities.</p>	<p>U_Q-10 Will the user have information about the expected outcome of performing undo at any given time (smart menu)?</p> <p>U_Q-11 If so, how will this information be provided to the user?</p>	<p>U_EX-5: Smart Menus in drawing app</p> <p>When the last performed operation in a drawing program was "paint red", the undo menu, or equivalent, should display "undo paint red" as opposed to the more generic "undo"</p>
<p>U_HCI-6 Providing object Specific Undo</p> <p>The software system must provide the possibility for the user to easily access (for example, through the right button) the specific commands that affect such an object. One should be the undo/redo function. In this case, the system should filter the global undo stack and show only the operations that affected the state of the selected object [36].</p>	<p>U_ELAB-6: Object-specific &amp; Global Undo</p> <p>Redo will only be available at object level if it is available globally. Same with undo.</p>	<p>U_Q-12 Will the system require Object Specific Undo?</p> <p>U_Q-13 If so, which system elements will require object-specific undo/redo?</p> <p>U_Q-14 How will this feature be accessed by the user?</p>	<p>U_EX-6: UML Design Program</p> <p>In a UML design program, selecting the graphic representation of a Class within a diagram would should provide the option to undo the operations performed on (and only on) this particular Class.</p>

### 5.2.1.2 Usability Elicitation Clusters

The Usability Elicitation Guideline in Table 5.2-1 suggests fourteen discussion items (U\_Q-1 to U\_Q-14) to be held with stakeholders in order to elicit all aspects of the Undo Functional Usability Feature. These discussion items can be clearly divided into seven initial groups, or clusters, as described in the Usability Elicitation Clusters in shown in Figure 5.2-1, according to the portion of the Undo functionality that they cover.

**U\_EC-1 Undoing actions:** The discussion items in this cluster, shown in Figure 5.2-1, cover which actions will need to be undone within the system for being considered ‘damaging’. For non-undoable actions (those that are considered ‘damaging’ but are determined to not be undoable) a warning of some kind may be required, deferring the discussion to the Warning Feature.

**U\_EC-2 Providing access to Undo:** Following the discussion flow (arrows) onto the next cluster in Figure 5.2-1, it can be observed that once it has been determined that Undoing actions will be required, and its details have been specified, this elicitation cluster discusses how the option to undo will be presented to the user.

**U\_EC-3 Redoing actions:** Once the need for an Undo functionality has been established, this elicitation cluster will determine if the Redo functionality is needed.

**U\_EC-4 Providing access to Redo:** This cluster embodies the discussion items regarding the way in which the Redo option will be presented to the user.

**U\_EC-5 Supporting Multi-level undo and history:** This cluster covers three discussions concerning the history log. It determines whether the history will be comprised of multiple levels, thus providing the possibility of undoing multiple actions as opposed to just the last one. Also, the possibility of accessing this history for purposes other than undoing/redoing actions (i.e. viewing them) is addressed.

**U\_EC-6 Providing expected results of Undo/Redo (“smart menus”):** Also related to the history log is the capability of providing ‘smart menus’ within the application, addressed in this cluster. When a user selects the option to ‘undo’ an action within an application, if the menu item is of the form of ‘Undo <action name>’ it is considered to be a ‘smart menu’ (the name of the action to undo is retrieved from the history log.)

**U\_EC-7 Undoing/Redoing actions over specific objects:** Lastly, this cluster covers the need of providing the user with the ability to undo the latest action performed over a specific object within the system.

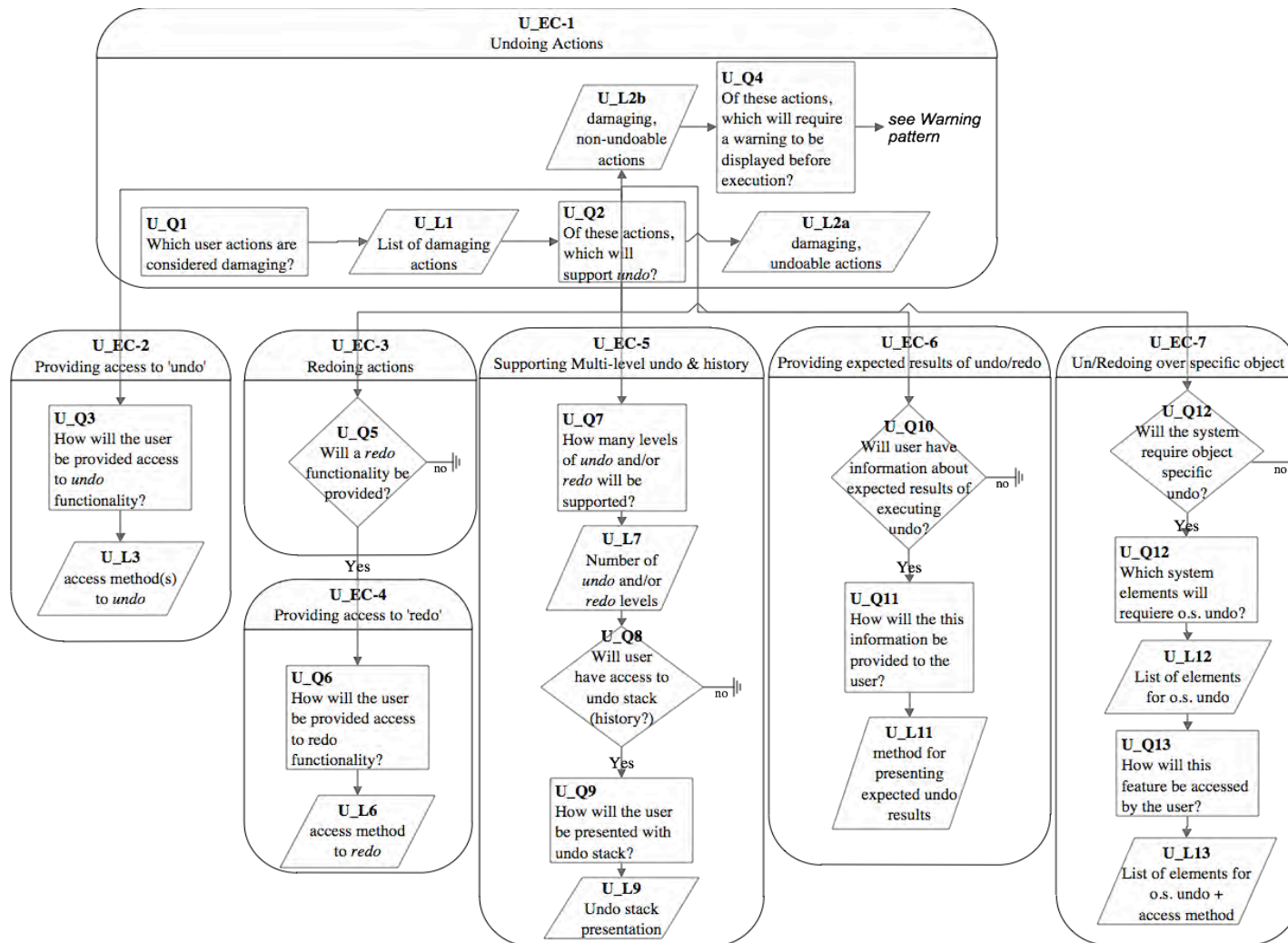


Figure 5.2-1: Elicitation Clusters. Undo.

### 5.2.1.3 Use Case Meta-model

The Use Case Meta-model for the Undo feature is shown in Figure 5.2-2 (See page 71 for color legend), in which seven use cases are identified and described below.

**U\_UC-1 Undo:** the user requests that the system revert the effects of the last invoked action. This can be done globally or by object. When doing it globally, the user calls Undo with no further specification, meaning that the last undoable action will be reverted. When doing it by object, Undo will be called over the object, which will revert the effects of the last undoable action that was invoked on it.

**U\_UC-2 Redo:** the user requests that the system revert the effects of the latest undo invoked, be it global or object-specific.

**U\_UC-3 ShowHistory:** The user request that the system display all the elements in the undo/redo queue (if any). This queue, or History (see U\_UC-7) holds the sequence of undoable actions that have been executed

**U\_UC-4 ShowNextUndoable:** The user requests (through ‘smart menus’, for example) that the system inform them of the name of the action that would be undone if Undo were to be executed at that specific moment.

**U\_UC-5 ShowNextRedoable:** The user requests (through ‘smart menus’ for example) that the system inform them of the name action that would be redone if Redo were to be executed at that specific moment.

**U\_UC-6 UndoableUserAction:** This use case is depicted in gray to illustrate that it is a “Template Use Case” and must be replaced by the actual action to undo/redo, if known (see page 71 for notation). The user orders the execution of an undoable action within the system. This use case represents the many undoable actions that a particular system can support. These actions can be either invoked by the user directly, or by the undo or redo commands when undoing or redoing, respectively. For example, the user can directly call an action called “turn light #5 on”, or it can be triggered as part of the invocation of Undo when the user calls to undo an action such as “turn light #5 off”

**U\_UC-7 SaveToHistory:** Whenever an Undoable User Action is executed, it is saved to the undo queue or History. This use case is triggered by every execution of an undoable action, hence an included use case of UndoableUserAction.

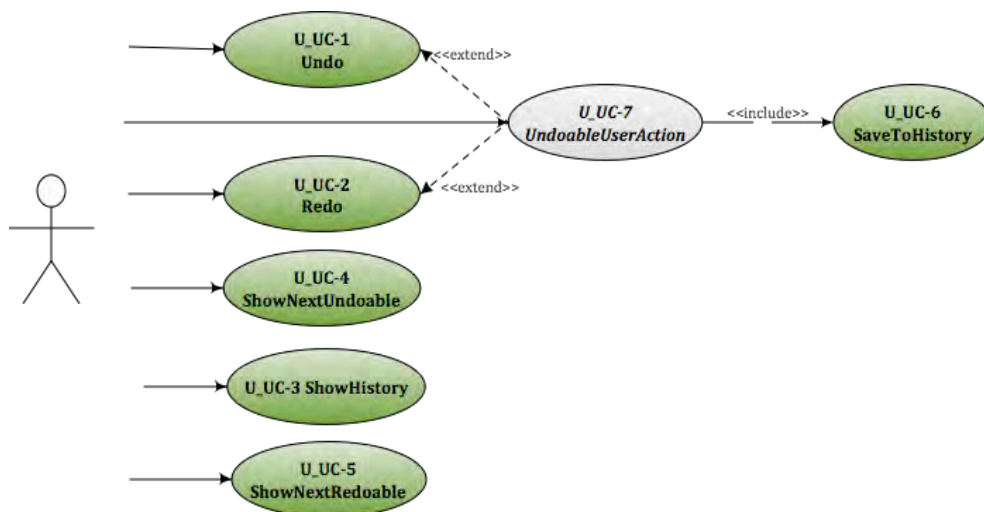


Figure 5.2-2 Use Case Meta-model. Undo

The applicability of each of these use cases will depend on the results of the elicitation process. If during elicitation of the Undo feature it is determined, for example, that no Redo feature is needed, then that use case will be discarded. Use cases also depend on one another. These dependencies are shown in Table 5.2-2, where we can see the following:

- The **Undo** use case needs the SaveToHistory use case, as for there to be anything to undo it must have been first saved to history. Similarly, it needs the UndoableUseCase (as does every other use case in this feature), which is the actual action to be undone.
- The **Redo** use case, aside from needing the SaveToHistory use case, needs the Undo use case. For there to be anything to redo, it must have been undone first.
- In the case of **ShowHistory**, aside from needing SaveToHistory, it needs the Undo use case. For a history list to make sense, actions need to be undone and stored in it.
- **ShowNextUndoable** and **ShowNextRedoable** have the same dependencies as ShowHistory.
- The **UndoableUserAction** needs the Save To History use case, because by definition, if it's undoable, it must be stored in history in order to be potentially undone.
- Finally, the **SaveToHistory** use case need the Undo use case, as if there are no actions being undone there will never be anything to store in the history list.

Table 5.2-2 Usability Use Case Dependencies: Undo Functional Usability Feature

	U_UC-1 Undo	U_UC-2 Redo	U_UC-3 Show History	U_UC-4 Show Next Undoable	U_UC-5 Show Next Redoable	U_UC-6 Undoable Usr Action	U_UC-7 Save to History
U_UC-1 Undo	-					X	X
U_UC-2 Redo	X	-				X	X
U_UC-3 Show History	X		-			X	X
U_UC-4 Show Next Undoable	X			-		X	X
U_UC-5 Show Next Redoable	X	X			-	X	X
U_UC-6 Undoable User Action						-	X
U_UC-7 Save to History	X					X	-

Looking at the columns of Table 5.2-2, both Undo and SavetoHistory are core to this feature. If either were discarded no other part of it could be implemented. Furthermore, implementing these two use cases alone would represent the minimal expression of the Undo feature, i.e. a system where the user can undo actions (stored in an internal list), and nothing more.

#### 5.2.1.4 System Responsibilities for Usability

Table 5.2-3 shows the proposed System Responsibilities for Usability for the present feature.

Table 5.2-3 System Responsibilities List for Undo

System Responsibilities List for Undo
U_SR-1 Store Undoable Executed Actions: The system must store all undoable actions that are executed
U_SR-2 Support Undo Functionality The system must allow users to undo executed (undoable) actions
U_SR-3 Provide Access to Undo The system must provide access to the Undo feature as agreed with user
U_SR-4 Support Redo Functionality The system must allow users to redo undone actions
U_SR-5 Provide Access to Redo The system must provide access to the Redo feature as agreed with user
U_SR-6 Support Multi-level Undo The system must allow undoing of as many executed actions as agreed
U_SR-7 Support Multi-level Redo The system must allow redoing of as many undone actions as agreed
U_SR-8 Show History The system must allow consulting the contents of the history of executed actions
U_SR-9 Provide Expected Results of Undo The system must provide smart menu functionality for Undo feature
U_SR-10 Provide Expected Results of Redo The system must provide smart menu functionality for Redo feature
U_SR-11 Provide Object-specific Undo The system must provide means to undo latest action(s) over an object
U_SR-12 Provide Object-specific Redo The system must provide means to redo latest action(s) over an object

These System Responsibilities for Usability are derived from the Usability Elicitation Clusters identified in section 5.2.1.2 as follows:

**U\_EC-1 Undoing actions:** As mentioned earlier, this elicitation cluster contemplates the stakeholder discussion items regarding the very basics of incorporating the undo functionality into any given system: firstly, will it be needed, and, if so, for which system actions will it be provided. This translates into the very core of the undo feature, which contemplates somehow saving such actions when they are executed, and eventually undoing them. Thus, this elicitation cluster would yield two such System Responsibilities, namely **U\_SR-1 Store Undoable Executed Actions** and **U\_SR-2 Support Undo Functionality**.

**U\_EC-2 Providing access to Undo:** This smaller elicitation cluster covers the details of how the undo functionality will be presented to the user (i.e. menu item, control sequence, etc) and thus yields the corresponding System Responsibility **U\_SR-3 Provide Access to Undo**.

**U\_EC-3 Redoing actions:** This elicitation cluster covers the need for a Redo feature for the actions chosen as undoable in U\_EC-1. It yields **U\_SR-4 Support Redo Functionality**.

**U\_EC-4 Providing access to Redo:** This elicitation cluster tackles the details regarding how the redo functionality will be presented to the user. In an analogous fashion to its redo counterpart, this cluster yields System Responsibility **U\_SR-5 Support Redo Functionality**.

**U\_EC-5 Supporting Multi-level Undo and Redo:** This elicitation cluster covers all aspects related to the stack or history unto which undoable actions will be stored for later undoing/redoing. As such it yields two System Responsibilities regarding the support for traversing this history while undoing/redoing, **namely U\_SR-6 Support Multi-level Undo** and **U\_SR-7 Support Multi-level Redo**. Furthermore, it covers the potential user need to not only traverse this list of executed actions, but to see it, hence yielding **U\_SR-8 Show History List**.

**U\_EC-6 Providing expected results of Undo/Redo:** This Elicitation Cluster covers the need for 'smart menus', where upon calling either Undo or Redo, the user will be informed of the next action to be undone/redone, respectively. It yields the two corresponding System Responsibilities **U\_SR-9 Provide Expected Results of Undo** and **U\_SR-10 Provide Expected Results of Redo**.

**U\_EC-7 Undoing/Redoing actions over a specific object:** This Elicitation Cluster resembles those pertaining to defining the Undo and Redo operations within the system, with the sole difference that in this case these operations are to be applied over specific objects. The discussions involved will provide as output a set of system actions (if any) that will require Undo and/or Redo. It yields two System Responsibilities: **U\_SR-11 Provide Object-specific Undo** and **U\_SR-12 Provide Object-specific Redo**.

Table 5.2-4 maps the relationships between the Usability Elicitation Clusters and the Usability System Responsibilities described above, for easy reference. Any project determined to require a specific Elicitation Cluster will also require its related System Responsibilities. Likewise, if a cluster is discarded during elicitation, its related responsibilities will not be a part of the resulting system.

Table 5.2-4 Usability Elicitation Clusters / System Responsibilities for Usability Mapping for Undo

Elicitation Clusters	System Responsibilities for Usability
U_EC-1 Undoing actions	U_SR-1 Store Undoable Executed Actions U_SR-2 Support Undo Functionality
U_EC-2 Providing access to Undo	U_SR-3 Provide Access to Undo
U_EC-3 Redoing actions	U_SR-4 Support Redo Functionality
U_EC-4 Providing access to Redo	U_SR-5 Provide Access to Redo
U_EC-5 Supporting Multi-level Undo and Redo	U_SR-6 Support Multi-level Undo U_SR-7 Support Multi-level Redo U_SR-8 Show History
U_EC-6 Providing expected results of Undo/Redo	U_SR-9 Provide Expected Results of Undo U_SR-10 Provide Expected Results of Redo
U_EC-7 Undoing/Redoing actions over a specific object	U_SR-11 Provide Object-specific Undo U_SR-12 Provide Object-specific Redo

## 5.2.2 Usability Guideline for Software Development: Design artifacts

The Undo feature is perhaps the most widely known Functional Usability Feature we cover in this work. It has been contemplated in literature for the past three decades and since then, numerous approaches for an architectural solution have emerged.

In the realm of object orientation, Gama et al. [37] propose what many OO languages today use at the core of their various implementations of the Undo feature: the ‘Command’ pattern.

In our work we have also used this pattern as the basis of the design artifacts for the Undo feature. It is of little contribution to ‘reinvent the wheel’ in this regard, so we’ve opted to adapt a pattern already proven successful to our specific needs: the use of an MVC architecture and additional HCI considerations not covered by the original Command pattern.

In section 5.2.2.1 the System Responsibilities are brought to a lower abstraction level as High-level Design Component Responsibilities, and section 5.2.2.2 as Low-level Design Component Responsibilities (for a MVC architecture). Finally, section 5.2.2.3 presents the Usability Design Meta-models for said Low-level Design Component Responsibilities as object-oriented class and sequence diagrams.

### 5.2.2.1 High-level Design Component Responsibilities

In order to support the System Responsibilities identified 5.2.1.4 at design level, the following sections describe the suggested High-level Design Components and their responsibilities for the Undo feature.

As mentioned earlier, the Undo feature has been addressed extensively in previous works. As such, those research results have already addressed significant parts of its expected functionality. Such is the case of the Command Pattern by the "Gang of Four" (GoF) [37], which we have chosen as the core of the design artifacts for this feature. Components (and, in the following sections, classes and objects) like the “Command” component, are taken directly from this widely known design pattern to fulfill the needs that arise from the corresponding System Responsibilities.

A smaller portion of the High-level Design Component Responsibilities described in the following sections, like some contained in the HistoryList and, further down, those of the History Exceptions, are not addressed by the original GoF Command pattern, and so are included as part of the original contribution of this work in order to fulfill the entirety of the expectations of this feature.

Another pattern used across all the guidelines is the GoF Observer pattern, as it is an integral part of the MVC architecture itself.

When represented graphically, objects and/or classes belonging to the GoF Command pattern or the Observer pattern are depicted in yellow and red, respectively (see page 71).

#### 5.2.2.1.1 User Interface (UI) Component

This component is responsible for capturing all user invocations and forwarding them (possibly through a delegating component) to the appropriate part of the domain, usually that responsible for executing the invoked action. Specifically for this feature, calls to undo or redo an action (globally or over a specific object) and calls to access the history list are captured by the UI Component.

The UI Component is also responsible for relaying information to the user, including appropriate feedback after an action has been executed. Within this feature, such information entails displaying ‘smart menus’ for the user and keeping them up-to-date every time an undo/redo is invoked.

For example, if the last action taken was italicizing a block of text, a ‘smart’ Undo menu would display as “Undo italics”. Once another action is taken within the application, for example, making a block of text bold, the ‘smart’ Undo menu should instantly change to “Undo bold”, keeping the user informed at all times (or whenever the Undo menu item is accessed) of the next undoable action.

Similarly, if the History List is visible to the user, the UI Component is responsible for keeping it up-to-date by ‘listening’ to all action invocations that may alter it.

#### 5.2.2.1.2 Domain Component

A Domain Component represents the part of the system that is responsible for executing the actions requested by the user. In an email program, for example, clicking the ‘Send’ button may have many intermediate effects (checking that the Subject field is not empty, loading attachments, etc), but the part of the system that is actually responsible for *sending* the email would be referred to as the Domain Component in the context of these features.

#### 5.2.2.1.3 Command Component

Based on Gamma’s definition of the Command Pattern [37], the Command Component is responsible for encapsulating method invocations and any pertinent state information at call-time.

When an action is invoked through the UI Component, it would normally be forwarded directly to the Domain Component responsible for executing said action. However, when the action that is called needs to be *undoable*, additional steps need to be taken before the Domain Component is allowed to execute it.

After the call is placed in the UI, a new instance of Command is created. This instance is initialized with the signature of the method that is being called and a reference to the Domain Component in charge of executing it, for later invocation. It is also initialized with any state information that will need to be restored if the invoked action is ever ordered to be undone.

Aside from storing this information, the Command Component is responsible for ultimately calling the invoked method in the Domain Command (once the state information has been saved), and for properly undoing said action (restoring the previous state information) should a call to do so come from the UI Component.

#### 5.2.2.1.4 History Component

The History Component is responsible for storing all Command instances ever created within the system. Once an undoable call is placed in the UI, the Command Component is created, as described previously, and immediately stored in the History Component. If a call to undo the latest action is placed in the UI, the History Component is responsible for locating (within itself) the Command instance which initially executed said action and ordering it to *undo*. The History Component is thus responsible for keeping a record of what the next undoable action is, both globally and for every specific object over which undo can be invoked.



Table 5.2-5: Usability Guideline: High-level Design Component Responsibilities. Undo feature.

System Responsibility	High-level Design Component Responsibilities
U_SR-1 Store Undoable Executed Actions	<p>The component responsible for handling user events (UI Component) must listen for calls to actions and order their execution</p> <p>Execution of actions is always the responsibility of the pertinent Domain Component in the application</p> <p>The component in charge of delegating actions (if any) should determine whether the action is undoable or not, from a pre-established list.</p> <p>If the action to execute is undoable, it must first be encapsulated as an instance of a Command Component, together with any pertinent state information and the necessary actions needed to revert its effects.</p> <p>Such an instance is then stored in a History Component, responsible for keeping a single (ordered) collection of all executed undoable actions.</p> <p>After encapsulation, the Domain Component is then free to execute the invoked action</p>
U_SR-2 Support Undo Functionality	<p>The UI Component must listen for calls to the Undo action (if available) and order its execution</p> <p>The History Component must then retrieve the last* Command, without discarding it, and order it to 'undo' itself.</p> <p>The Command, in turn, executes the necessary actions, using the stored state information, to return the system to the state preceding its execution</p>
U_SR-3 Provide Access to Undo	<p>The UI Component is responsible for providing the mean(s) through which a user can invoke the undo feature. The Undo action should only be available when at least one undoable action has been executed during application up-time</p>
U_SR-4 Support Redo Functionality	<p>The UI Component must listen for calls to the Redo action (if available) and order its execution</p> <p>The History Component must then retrieve the current** Command, without discarding it, and order it to 'redo' itself.</p> <p>The Command, in turn, executes the necessary actions, using the stored state information, to return the system to the state that follows its execution</p>
U_SR-5 Provide Access to Redo	<p>The UI Component is responsible for providing the mean(s) through which a user can invoke the redo feature. The Redo action should only be available when the Undo action has been executed at least once before during application up-time.</p>
U_SR-6 Support Multi-level Undo	<p>When only one-level undo is supported, the History component holds only the last-executed action. However, when multi-level undo is supported, History supports an ordered collection of said actions in the form of Commands.</p> <p>The History Component is also responsible for updating (or ordering the update of) the UI every time a new Command is added to the collection or whenever the next undoable action changes.</p>
U_SR-7 Support Multi-level Redo	<p>As is the case with supporting multi-level Undo, when multi-level redo is supported, History holds an ordered collection of actions in the form of Commands,</p> <p>The History Component is also responsible for updating (or ordering the update of) the UI every time a new Command is added to the collection or whenever the next redoable action changes.</p>
U_SR-8 Show History	<p>The UI is responsible for listening for updates from History Component</p> <p>The History Component must keep the View informed at all times of any changes to its contents (i.e. addition/deletion of Commands, etc)</p>
U_SR-9 Provide Expected Results of Undo	<p>Whenever the Undo actions is available, the UI Component is responsible for showing the actions' expected results (smart menus)</p> <p>The UI gets this information from the History Component, which must notify it of the current undoable action upon every change.</p>
U_SR-10 Provide Expected Results of Redo	<p>Whenever the Redo action is available, the UI Component is responsible for showing the actions' expected results (smart menus)</p> <p>The UI gets this information from the History Component, which must notify it of the current re-doable action upon every change.</p>
U_SR-11 Provide Object-specific Undo	<p>The UI Component must listen for calls to the Undo action over a particular object (if available) and order its execution.</p> <p>The History Component must then retrieve the last Command executed over that object, without discarding it, and order it to Undo itself.</p> <p>The Command, in turn, executes the necessary actions, using the stored state information, to return the object to the state preceding its execution.</p>
U_SR-12 Provide Object-specific Redo	<p>The UI Component must listen for calls to the Redo action over a particular object (if available) and order its execution.</p> <p>The History Component must then retrieve the current Command executed over that object, without discarding it, and order it to Redo itself.</p> <p>The Command, in turn, executes the necessary actions, using the stored state information, to return the object to the state following its execution.</p>

### 5.2.2.2 Low-level Design Component Responsibilities for MVC

When instantiating for an MVC architecture, the High-level Design Components described above can be translated into the following system objects.

The User Interface Component is instantiated by the **View** object(s) and takes over all of its responsibilities except for the delegation of actions to other objects. This responsibility falls on the **Controller** object(s) of the MVC architecture.

The Command Component is defined in the **Command** interface and implemented by **ConcreteCommand** objects, as specified in the GoF Command Pattern. For every undoable command there will exist a distinct ConcreteCommand class (i.e. EmptyTrashCommand, DeleteObjectCommand, etc.). Whenever a command is called through the View, the corresponding ConcreteCommand object will be created, saved to history and ordered to execute.

The History Component is represented by the **HistoryList** singleton class, and covers all the responsibilities of the History Component.

The Domain Component is represented by the **DomainClass**. This class is represented in gray in Figure 5.2-3 as a reminder that this is not an actual class nor does it fall within the scope of the proposed design model, but is rather an entity to be substituted at design time by the class that actually performs the requested task.

A hierarchy of Exception classes is also provided within this model (**HistoryException**, **UndoException**, **RedoException**, **NothingToUndoException**, **NothingToRedoException**, **NothingToUndoForObjectException** and **NothingToRedoForObjectException**). These provide error information during the execution of Undo and Redo, both globally and over specific objects.

Table 5.2-6 details the Low-level Design Component Responsibilities described above and how they carry out each of the System Responsibilities defined in section 5.2.1.4. For each System Responsibility, the sequence of actions required by the different objects is presented as well as a set of UML diagrams that depict each of these interactions.

Table 5.2-6: Usability Guideline: Low-level Design Component Responsibilities (MVC)

System Responsibility	Objects					Fig
	View	Controller	ConcreteCommand	HistoryList	DomainClass	
U_SR-1 Store Undoable Executed Actions	1. The <i>View</i> must listen for invocation of actions. Upon reception, it must notify the <i>Controller</i> of said action.	2. The <i>Controller</i> must determine if the invoked action is undoable. In such case it must call the <code>execute()</code> method of the corresponding <i>ConcreteCommand</i> object (otherwise invocation goes directly to the <i>DomainClass</i> ). 3. The <i>Controller</i> must then <code>clone()</code> said <i>ConcreteCommand</i> and <code>add()</code> it to the <i>HistoryList</i> .	4a. Upon call to its <code>execute()</code> method, the <i>ConcreteCommand</i> first stores the necessary state information in its local variables. It then calls the appropriate method in the corresponding <i>DomainClass</i> (what was originally invoked).	4b. The <i>HistoryList</i> saves the cloned <i>ConcreteCommand</i> atop its collection (so it can later be available to undo).	5a. The <i>DomainClass</i> executes the appropriate method to carry out what was originally invoked by the user through the <i>View</i> .	Figure 5.2-4
U_SR-2 Support Undo Functionality	1. The <i>View</i> must listen for invocation of the Undo action. Upon reception, it must notify the <i>Controller</i> .	2. The <i>Controller</i> orders the <i>HistoryList</i> to undo the last action.	4. Upon call to its <code>undo()</code> method, the <i>ConcreteCommand</i> calls the necessary methods in <i>DomainClass</i> (with any needed state information, stored upon execution) to revert its effects.	3. The <i>HistoryList</i> determines the <i>ConcreteCommand</i> to undo and calls its <code>undo()</code> method.	5. The <i>DomainClass</i> executes the methods invoked by <i>ConcreteCommand</i> .	Figure 5.2-5
U_SR-3 Provide Access to Undo	1. The <i>View</i> must present the user with the mean(s) to call the Undo action (i.e. within the Edit menu, through Ctrl-Z, etc.)					Figure 5.2-5
U_SR-4 Support Redo Functionality	1. The <i>View</i> must listen for invocation of the Redo action. Upon reception, it must notify the <i>Controller</i> .	2. The <i>Controller</i> orders the <i>HistoryList</i> to redo the current action.	4. Upon call to its <code>redo()</code> method, the <i>ConcreteCommand</i> calls the necessary methods in <i>DomainClass</i> (with any needed state information, stored upon execution) to reinstate its effects.	3. The <i>HistoryList</i> determines the <i>ConcreteCommand</i> to redo and calls its <code>redo()</code> method.	5. The <i>DomainClass</i> executes the methods invoked by <i>ConcreteCommand</i> .	Figure 5.2-6
U_SR-5 Provide Access to Redo	1. The <i>View</i> must present the user with the mean(s) to call the Undo action (i.e. edit menu, Ctrl-Z, etc.)					Figure 5.2-6
U_SR-6 Support Multi-level Undo	3. When the <i>View</i> is notified of changes in the <i>HistoryList</i> it updates its History Displays accordingly.			1. The <i>HistoryList</i> stores (clones of) <i>ConcreteCommands</i> in a FILO-ordered collection. It keeps a 'pointer' of the last action invoked, and moves it back every time Undo is called until no more <i>ConcreteCommands</i> exist. Invoking Redo moves the 'pointer' forward in a similar fashion. <i>ConcreteCommands</i> are never removed from the <i>HistoryList</i> , except when its maximum allowed size is reached (in which case the older elements will be removed in order). 2. Every time a <i>ConcreteCommand</i> is added to the <i>HistoryList</i> or the 'pointer' changes position (i.e. the next undoable/re-doable action is updated), the <i>HistoryList</i> notifies the <i>View</i> .		Figure 5.2-5 Figure 5.2-6

System Responsibility	Objects					Fig
	View	Controller	ConcreteCommand	HistoryList	DomainClass	
U_SR-7 Support Multi-level Redo	2. When the View is notified of changes in the HistoryList it updates its next un/re-doable.			1. Every time a ConcreteCommand is added to the HistoryList or the 'pointer' changes position (i.e. the next undoable/re-doable action is updated), the HistoryList notifies the View		Figure 5.2-6
U_SR-8 Show History	1. The View must subscribe to and listen for updates from the HistoryList at all times.			2. Every time HistoryList modifies its contents (by adding a new ConcreteCommand or by ordering an existing one to undo, thus changing the position of the pointer) it must notify the View		Figure 5.2-4 Figure 5.2-5 Figure 5.2-6
U_SR-9 Provide Expected Results of Undo	2. The View receives the notification and updates the text for the next undoable item			1. Upon undoing, the HistoryList notifies the View of its new structure (same as previous minus top action)		Figure 5.2-4 Figure 5.2-5
U_SR-10 Provide Expected Results of Redo	2. The View receives the notification and updates the text for the next redoable item			1. Upon redoing, the HistoryList notifies the View of its new structure (same as previous except for new top action)		Figure 5.2-4 Figure 5.2-6
U_SR-11 Provide Object-specific Undo	1. The View must listen for invocation of the Undo action over a specific object. Upon reception, it must notify the Controller.	2. The Controller orders the HistoryList to undo the last action over object.	4. Upon call to its <code>undo()</code> method, the ConcreteCommand calls the necessary methods in DomainClass (with any needed state information, stored upon execution) to revert its effects.	3. The HistoryList determines the ConcreteCommand to undo by filtering for that particular object, and calls its <code>undo()</code> method.	5. The DomainClass executes the methods invoked by ConcreteCommand.	Figure 5.2-4 Figure 5.2-5
U_SR-12 Provide Object-specific Redo	1. The View must listen for invocation of the Redo action over a specific object. Upon reception, it must notify the Controller.	2. The Controller orders the HistoryList to redo the last action over object.	4. Upon call to its <code>redo()</code> method, the ConcreteCommand calls the necessary methods in DomainClass (with any needed state information, stored upon execution) to restore what was undone.	3. The HistoryList determines the ConcreteCommand to redo by filtering for that particular object, and calls its <code>redo()</code> method.	5. The DomainClass executes the methods invoked by ConcreteCommand.	Figure 5.2-4 Figure 5.2-6

### 5.2.2.3 Usability Software Design Meta-models

This section describes the UML diagrams representing the Low-level Design Component Responsibilities described in Table 5.2-6. Below, the class diagram for this feature is presented, along with a short description of the classes involved and their interrelationships. This is followed by the sequence diagrams for this feature, representing the sequences explained earlier in (see Table 5.2-6).

#### 5.2.2.3.1 Class Diagram

Figure 5.2-3 below shows the class diagram for the Undo Functional Usability Feature. As described in the Low-level Design Component Responsibilities Table (see Table 5.2-6), the main objects involved are the View, Controller, HistoryList, ConcreteCommand and DomainClass. Two additional elements can be seen in the class diagram and are explained below: The Command Interface (as implemented by the GoF [37]) and the HistoryException (and children) to handle all possible errors.

The View and the Controller, fulfilling their role within MVC, respectively capture and distribute the user calls to perform actions. The HistoryList controls access as well as storage for the list of actions (or commands) that have been executed and stored for potential undoing.

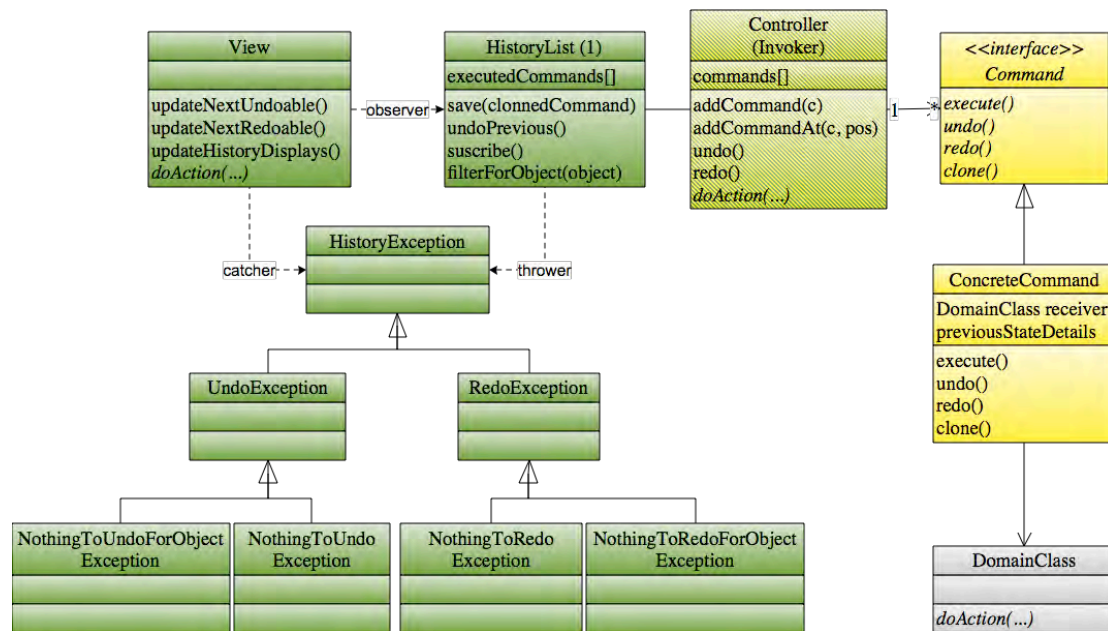


Figure 5.2-3: Usability design model. Class diagram. Undo.

The ConcreteCommand class implements the Command interface, and is responsible for ordering the execution of the requested action (in DomainClass) as well as for storing all necessary state information required for eventually undoing the command it represents (i.e. the method it is calling within DomainClass). In any given system there will be as many ConcreteCommands as there are undoable actions [37], and it is recommended they be labeled by an appropriate mnemonic. For example, the ConcreteCommand class in charge of invoking the sendMail() method in an email application should be labeled something like SendMailCommand or SendMailConcreteCommand, etc.

These classes and interfaces which belong to the Command Pattern of the GoF [37] are depicted in yellow (see page 71) to differentiate them from the rest of the classes particular to this meta-model.

It's worth noting that the Controller class also includes the Invoker functionality specified by the GoF Command Pattern and is thus depicted as partly belonging to this pattern (half green for the Undo FUF, half yellow for the GoF Command pattern). Similar is the case of HistoryList which covers both original functionality and functionality belonging to the GoF Command pattern.

Lastly, the family of exceptions is defined, rooted at HistoryException. This is the kind of exception that will be thrown in case an error occurs during undo/redo operations. For undo operations, an exception of the type UndoException will be thrown, and, likewise, for exceptions occurring redo operations, RedoExceptions will be thrown. UndoExceptions as well as RedoExceptions can be thrown for a specific object (when trying to undo the last action performed over a particular object) or for the application as a whole. For the undo operation, the concrete exceptions that will be thrown and eventually captured and handled by the View are NothingToUndoException, when the user is invoking the undo method when there's actually no actions to undo, NothingToUndoForObject, for a particular object, and, similarly, NothingToRedoException and NothingToRedoForObject for the redo operation. These exceptions will be thrown by HistoryList when attempting these four illegal operations.

#### 5.2.2.3.2 Sequence Diagram "Executing"

Figure 5.2-4, shows the sequence diagram for calling, executing and storing undoable actions.

This sequence starts when the user requests an undoable action to be executed. Following the GoF Command Pattern [37], prior to execution, the ConcreteCommand which encapsulates the invoked action (doAction()) is cloned (preserving its state information intact) and stored, in this case in the HistoryList object. After that, it is ordered to execute(), which simply entails a call to the desired method of the corresponding DomainClass. After execution, the View updates its display of the History List and updates the 'next undoable' field, if applicable. When the invoked action is not undoable (second part of *alt* in Figure 5.2-4), the desired method is invoked directly off DomainClass and executed.

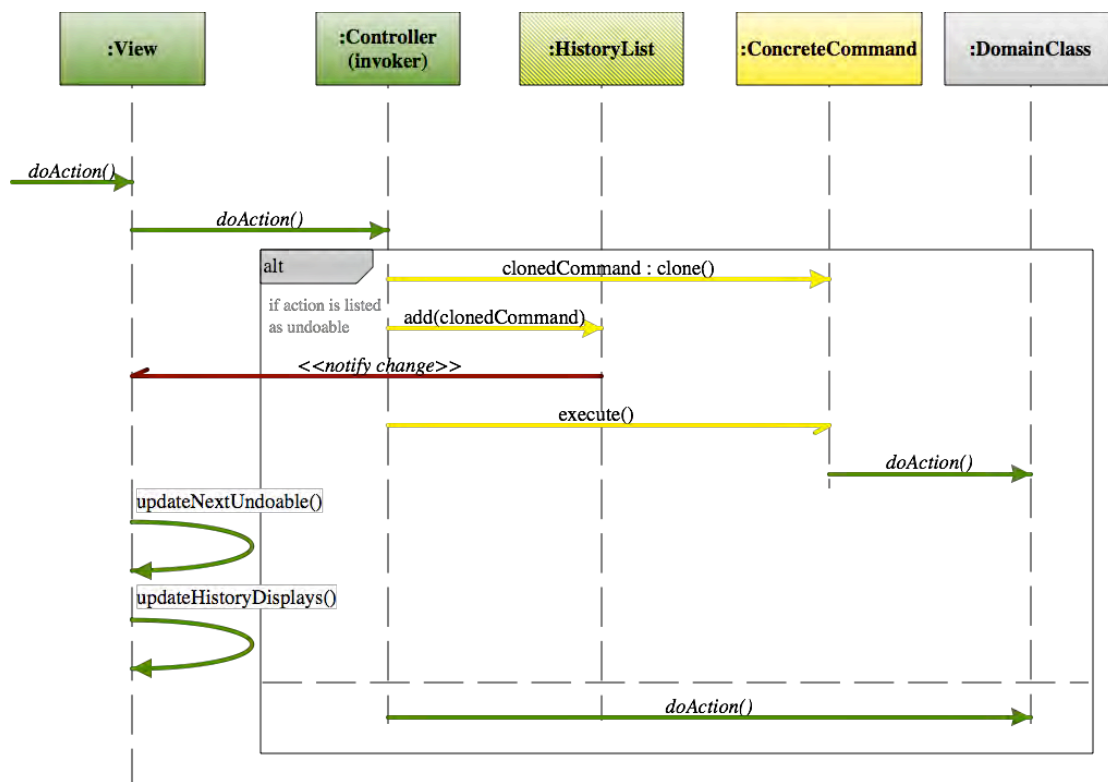


Figure 5.2-4: Sequence Diagram "Executing". Undo.

Classes and methods depicted in green represent they belong to the Undo FUF. Those in yellow are part of the GoF Command Pattern and notifications in dark red represent Observer Pattern functionality [37]. The gray DomainClass is a template class to be substituted at design time by the appropriate system class containing the undoable action.

### 5.2.2.3.3 Sequence Diagram “Undoing”

Figure 5.2-5 shows the sequence diagram for undoing actions.

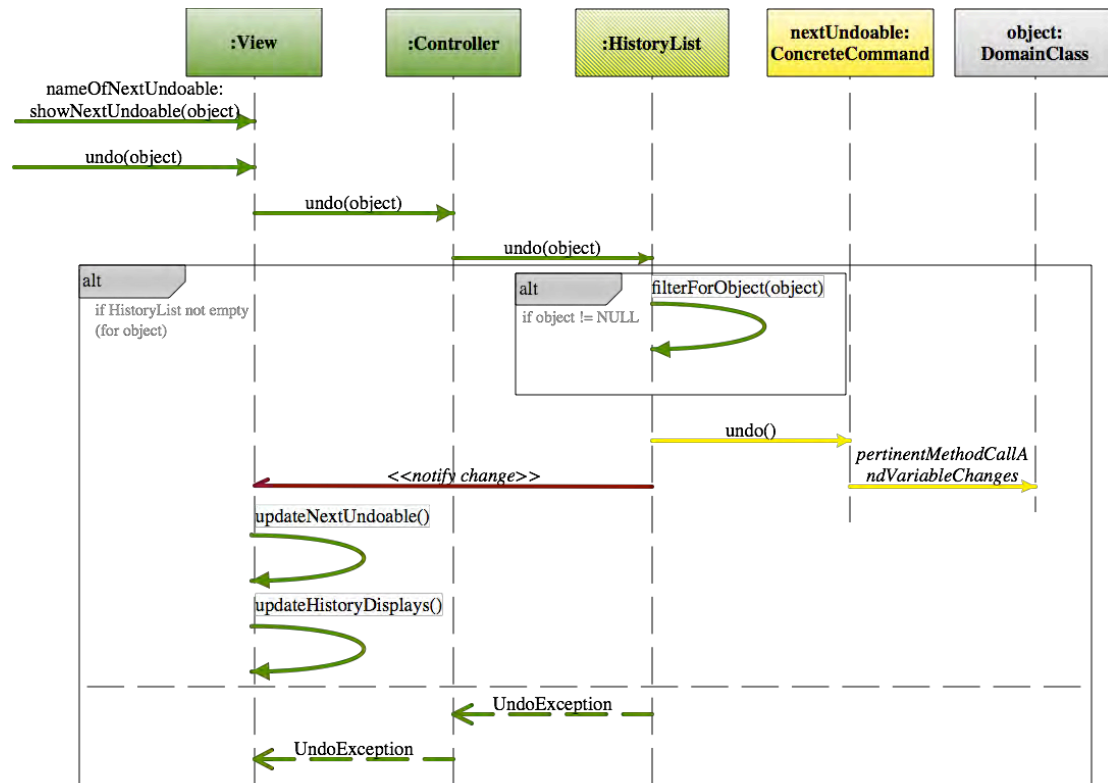


Figure 5.2-5: Undo. Sequence Diagram “Undoing”

The sequence starts off when the user requests to undo the next available action. It does so by asking the View what the name of the next undoable action is (i.e. by choosing on the “Edit - > Undo <action name>” menu item) followed by the actual invocation of undo over said action (i.e. clicking on the menu item).

The request to undo can be invoked over a specific object (see *object* parameter of *undo()* method below) or, by passing no parameter to the method, it is invoked over the application as a whole. Either way, the request to undo is passed on to the Controller, which in turn chooses the History list as the appropriate class to handle the request.

If *undo* is being invoked over an *object*, HistoryList finds the next available undoable action for that particular object with *filterForObject(object)*, otherwise, the next undoable action is always the one on top of the History List. Either way, the object returned is of the type ConcreteCommand, and it represents the action to undo.

As specified by the GoF Command pattern, the *undo()* method in the ConcreteCommand is then invoked, triggering a call (or series of calls) to the appropriate methods in the corresponding DomainClass in order to reinstate the previous state information, effectively undoing the action.

Upon returning said ConcreteCommand, the HistoryList effectively changes as the action on top (the next undoable action) has changed. This triggers a notification from the HistoryList

to all of its listeners (GoF Observer pattern). In this case, the only relevant listener, and complying with MVC, is the View. As such, the View gets notified when the HistoryList changes. This notification prompts the View to update the GUI information about the next undoable item (the text in the Edit -> Undo menu) and that of any history list displays it may maintain.

If the initial call to undo upon HistoryList is met with an empty list (or, in the case of object-specific undo, with a HistoryList containing no actions for that particular object), an exception of the type UndoException will be returned and forwarded to the View. The concrete exception returned will be NothingToUndoException in the case of having invoked undo over the entire application, and NothingToUndoForObjectException in case of a concrete object.

#### 5.2.2.3.4 Sequence Diagram "Redoing"

Figure 5.2-6 shows the sequence diagram for redoing actions. It is analogous to the sequence diagram for Undoing described for "undoing" in the sequence of events and objects involved, except where in the previous sequence the undo method was called, in this sequence the redo method is called, both over an object or over the application itself.

An empty HistoryList or one with no actions to redo over a particular object (in the case of object-specific redo) will return a RedoException of the appropriate type, similarly to the case of undoing.

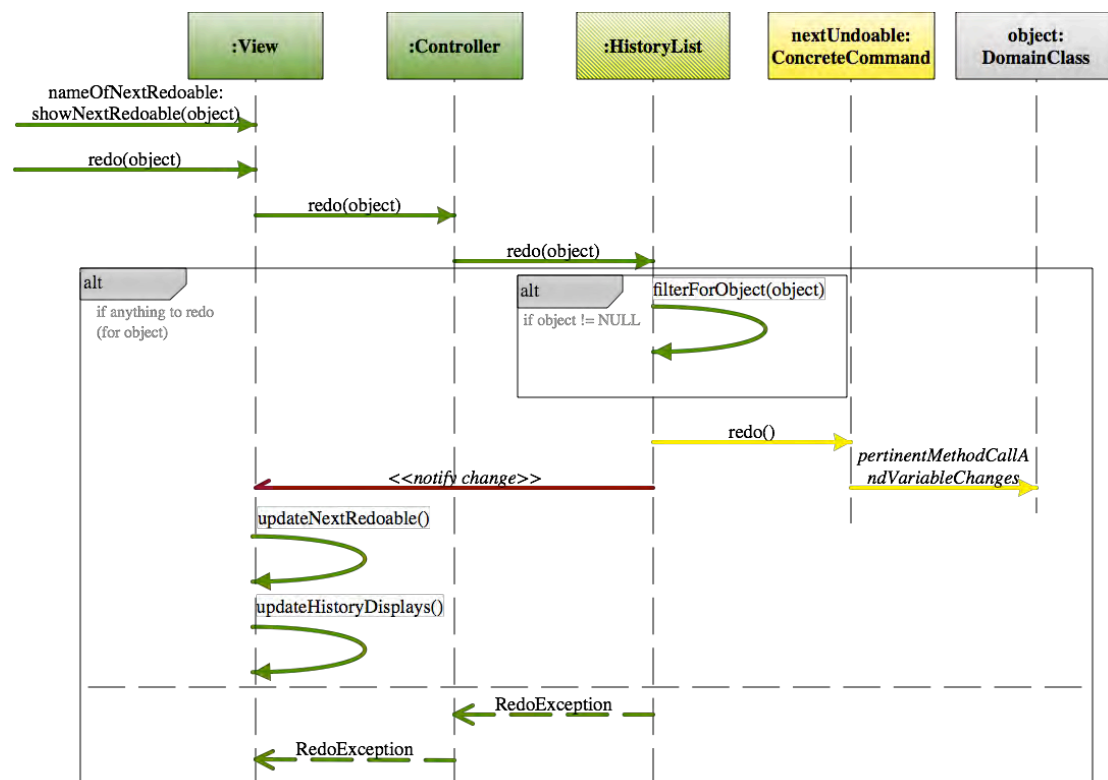


Figure 5.2-6: Sequence Diagram "Redoing". Undo.



## 5.3 “Abort” Usability Guideline for Software Development

The Abort Functional Usability Feature covers the user’s need to cancel on-going commands within an application and to exit the application altogether. Certain commands might take a long time to execute. In such cases, the user will need to be at the liberty to abort them. Furthermore, they must also be allowed to exit the application at all times in a stable way, regardless of any tasks that may be being executed.

The Usability Guideline for Software Development is made up of Analysis artifacts and Design artifacts. These are described for the Abort feature in the following two sections.

### 5.3.1 Usability Guideline for Software Development: Analysis artifacts

There are four artifacts to be used during the analysis phase: the Usability Elicitation Guideline, the Usability Elicitation Clusters, the Usability Use Case Meta-models, the System Responsibilities for Usability. These artifacts are described in the following four sections.

#### 5.3.1.1 Usability Elicitation Guideline

Table 5.3-1 shows the Usability Elicitation Guideline for the Abort feature. In this guideline there are two HCI recommendations described in detail below.

##### 5.3.1.1.1 Cancelling Commands

HCI authors suggest that commands that normally last over two seconds to execute should provide means for the user to cancel them (A\_HCI-1). The term ‘command’ is understood as an indivisible task being executed at any given time within the system, for which there are no viable partial results until execution is finished. (A\_ELAB-1). As such, all changes that may have occurred within the system due to the partial execution of said command be undone automatically. (A\_ELAB-2). Finally, the ways in which commands are usually cancelled within GUI-based application are fairly standard and only in cases where no standard applies should stakeholders debate the presentation for this feature (A\_ELAB-3)

Stakeholders must determine which commands are likely to last over two seconds, thus needing to have a cancel option. The way this cancel option is to be presented to the user, or any exceptions (long commands that simply cannot or should not be cancelled) must also be specified, as well as how to deal with any changes that may have occurred within the system due to the partial execution of said command (A\_Q-1 to A\_Q-3). In Table 5.3-1, example A\_EX-1 “Exporting video file” describes an example for this HCI recommendation.

##### 5.3.1.1.2 Exiting the Application

The Final HCI recommendation for the Abort feature contemplates exiting the application altogether. Said option should always be available to the user, regardless of any commands they may be executing at any given time. If any data has changed at exit time, the user must be prompted to save said changes (A\_HCI-2).

When exiting an application, it is not only data changes that must be considered, but also the repercussion of said exit upon any running commands. Upon elicitation time, stakeholders must discuss whether, a) any running commands will be aborted (and their effects rolled back), b) the application will wait for any ongoing commands and only exit after they have all have finished, c) the application will defer the decision to the user, asking them if they’re sure they want to exit, and by doing so cancel all ongoing commands, or if they would rather wait for them to finish (and then have the application exit on its own). (A\_ELAB-2 to A\_ELAB-4). Discussions A\_Q-4 to A\_Q-6 deal with how the exit will be handled regarding the aforementioned options. In Table 5.3-1, example A\_EX-2 “Apple Mail Exit” describes an example for this HCI recommendation, when the application uses option (b) as the standard way of exiting.

Table 5.3-1. Usability Requirements Elicitation Guideline. Abort.

Identification			
Name	Abort		
Family	Undo/Cancel		
Aliases	Emergency Exit [11]; Go Back to a Safe Place [42]; Go Back [42]; Prominent Cancel [42]		
Intent			
Providing the means to cancel an on-going task, or to allow for exiting the application altogether			
Problem			
Certain tasks might take a long time to execute. In such cases, the user will need to be at the liberty to cancel them. S/he must also be allowed to exit an application at all times, regardless of any tasks that may be being executed.			
Context			
When the user needs to exit an application or a command quickly.			
Interrelationships			
When implementing the Abort feature, Undo functionality will be needed for the cancellation of commands, in order for the application state to be properly reverted. Also, if implementing an application that prompts the user to save changes upon exiting, parts of the Warning feature will be needed.			
HCI Recommendation	Elaboration	Discussions with Stakeholders	Usage Examples (optional)
<p>A_HCI-1 Cancelling Commands</p> <p>If a command takes longer than 10 seconds to execute, there should be an option to cancel it, to allow interruption of its execution and return to the previous state. [38].</p>	<p>A_ELAB-1 Identification/Selection</p> <p>A "command" is an indivisible task invoked by the user, for which there are no viable partial results until execution is finished. It is necessary to identify commands that can be potentially long (&gt;10s) as per the HCI recs.</p> <p>In the case that listing all potentially long commands is not viable, only those requiring special attention should be listed, and a default cancellation behavior defined for the rest of them. Once identified, it must be determined if there are any that should, exceptionally, not be cancellable.</p>	<p>A_Q-1 Which commands will require a cancel option?</p> <p>A_Q-2 For all cancelable commands, how should the cancel option be presented to the user?</p> <p>A_Q-3 For all cancellable commands, which state will the system go to after the user chooses the cancel option?</p>	<p>A_EX-1 Exporting Video File</p> <p>In Apple's Quicktime, choosing the option to 'export' a video file into a different format, the application does so presenting a progress bar with a cancel button. Upon cancellation, any portion of the video that was exported is automatically sent to the 'Trash'.</p>
	<p>A_ELAB-2 Presentation</p> <p>There are standard simple ways of cancelling commands (i.e. button X, ctrl-c, etc.). Only in specific cases with particular needs should the presentation of the cancel option deviate from these standards</p>		
	<p>A_ELAB-3 Going back to previous state</p> <p>When cancelling a command it is necessary to automatically undo the effects that this command may have produced while executing.</p>		
<p>A_HCI-2 Exiting the application</p> <p>When users work with a high number of applications at the same time they may need to exit one of them quickly when a more important task may need the system resources, or when he started the application by mistake [11]. It is important to provide, at application level, that the option to exit the program is always clearly available (even</p>	<p>A_ELAB-4 Presentation of "Exit" Option</p> <p>The user must be consulted about any specific needs regarding the presentation of the Exit option. Otherwise, the standards offered by the OS or language should be used.</p>	<p>A_Q-4 Where and how will the exit option be presented to the user?</p> <p>A_Q-5 Will the user be presented with the option to 'save changes'?</p> <p>A_Q-6 If so, how will the user be presented with the option to 'save changes'?</p> <p>A_Q-7 If upon selection of 'exit' option there are running commands/operations, how will the system handle them (option a, b or c)?</p>	<p>A_EX-2 Apple Mail Exit</p> <p>When exiting Apple's Mail application, any outgoing emails that are pending are fully sent and only after this does the application (automatically) close.</p>
	<p>A_ELAB-5 Handling changes</p> <p>Generally, the option to "save changes" is relevant in applications that modify files during execution. An alert message indicating that there exist changes that need to be saved will be shown as a Warning of the type "confirmation" (see Warning feature) to the user</p>		

Identification			
<p>during program startup) and that this option will never be obscured by dialogue windows</p> <p>If the user chooses the option to Exit after having changed data, the application should prompt him to "save changes"</p>	<p>A_ELAB-6 Presentation of "Save Changes" option</p> <p>The user must be consulted about any specific needs regarding the presentation of the Save Changes option. Otherwise, the standards offered by the OS or language should be used.</p>		
	<p>A_ELAB-7 Ending commands upon exit</p> <p>There are three options regarding handling commands upon exit:</p> <p>a) Immediate Exit: Cancelling all running commands, discarding their results and closing the application without further consulting the user.</p> <p>b) Wait and Exit: Control of the application is given to the running commands. When the last of them is done executing, the application exits automatically.</p> <p>c) Prompt: Giving the user the option of choosing between "a" and "b" upon closing the application.</p>		

### 5.3.1.2 Usability Elicitation Clusters

The Usability Elicitation Guideline in Table 5.3-1 suggests seven discussions items (A\_Q-1 to A\_Q-7) to be held with stakeholders in order to elicit all aspects of the Abort Functional Usability Feature. According to the covered portion of the functionality, these discussion items can be grouped into four initial sets, or clusters, as described in the Usability Elicitation Clusters in shown in Figure 5.3-1.

**A\_EC-1 Identifying cancellable commands:** The discussion items in this cluster, shown in Figure 5.3-1, determine which commands will be cancellable depending on how long they usually take to execute.

**A\_EC-2 Cancelling commands and handling application state:** Once a list of cancellable commands (or *type* of commands if too extensive) has been determined, stakeholders must decide how said cancel option will be presented to the user and the state that the system will go to after cancelling each of these commands.

**A\_EC-3 Exiting application while handling potential on-going commands/operations:** These discussions will determine if and how the option to exit the application will be presented to the user. Once that has been established, it must be decided how to handle any potential ongoing commands or operations that may exist at exit time.

**A\_EC-4 Handling potential changes to be saved:** If the application is of the type that may need changes to be saved, it must be determined if such a feature must be presented to the user upon exiting the application when changes are pending.

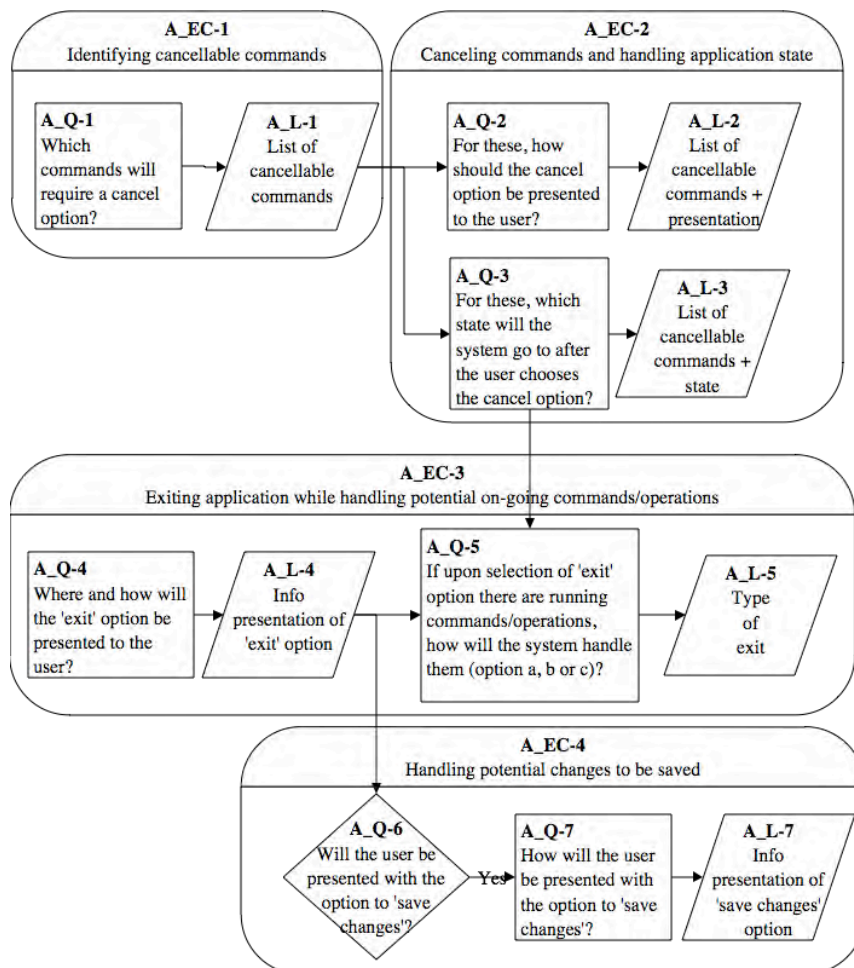


Figure 5.3-1: Elicitation Clusters. Abort feature.

### 5.3.1.3 Use Case Meta-model

Figure 5.2-2 shows the Use Case Meta-model for the Abort Functional Usability Feature. In this meta-model, four concrete use cases are identified (plus three borrowed use cases that apply) and are described in full below.

**A\_UC-1 CancelCommand:** The user chooses the option to cancel an ongoing command (the Long/Undoable User Action). Canceling an action imply undoing changes in which it may have incurred (see extension U\_UC-1 Undo).

**A\_UC-2 Exit:** The user chooses the option to exit the application. This action may prompt the user to save any pending changes (see extension: A\_UC-3 SaveChanges). It also may trigger the cancellation of any ongoing actions (whether they are commands or operations. See extension: A\_UC-1 Cancel)

**A\_UC-3 SaveChanges:** Upon exiting the application, the user is prompted to save any pending changes.

**A\_UC-4 Long/UndoableUserAction:** Any action invoked by the user which is considered ‘long’, needing some form of progress information to be shown (see included use case SPF\_UC-1 ShowProgress, and ‘undoable’, needing to be saved to the history log for potential undoing (see included use case U\_UC-6 SaveToHistory).

Borrowed use cases:

**SPF\_UC-1 ShowProgress:** Shows the progress of an ongoing action. See Progress Feedback.

**U\_UC-1 Undo:** Reverts the effects of an executed action. See Undo

**W\_UC-3 Confirm:** Prompts the user to proceed (OK) or cancel an action. See Warning

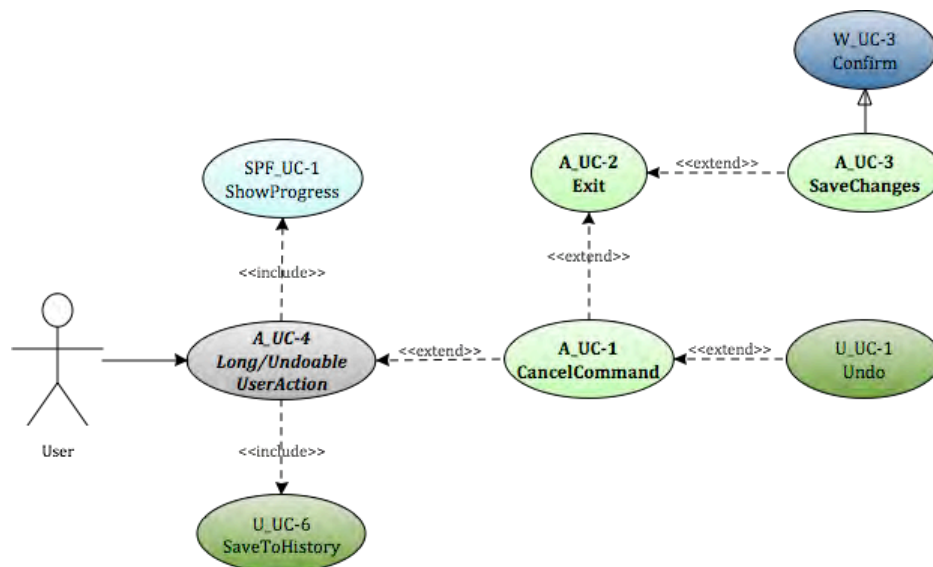


Figure 5.3-2 Use Case Meta-model. Abort

As mentioned above, whether or not all of these use cases apply for a given system will depend on the results of the elicitation process. If during elicitation of the Abort Functional Usability Feature it is determined that, for example, there will never be changes to save upon exiting the application, the SaveChanges use case, and any use cases that extend from it, will be discarded. Similarly, use cases depend on one another. These dependencies are shown in Table 5.3-2 for the Abort feature.

- The **Cancel** use case needs the **Long/UndoableUserAction** use case, as if there are no actions to cancel, the ‘cancel’ feature becomes irrelevant. Furthermore, this use case depends on **Undo** for undoing any changes the action being cancelled may have caused.
- The **Exit** use case can potentially need every use case in this meta model. If commands are considered for cancellation upon exit time, **CancelCommand** will be needed for Exit to be viable. If data changes are being monitored upon exiting the application and the user needs to be prompted for saving, then the **SaveChanges** use case will be needed. Furthermore, if/when considering the Cancel use case, the **Long/UndoableUserAction** use case must be considered, as it represents the action to revert upon cancellation. Finally, for any undoing to take place, if applicable, the **Undo** use case will be needed.
- Likewise, when exiting does not require checking for ongoing commands or changes, the **Exit** use case will have none of the aforementioned potential (asterisked) dependencies.
- The **SaveChanges** use case depends on the **Exit** use case existing, as it is devised as a prompt for the user to save changes *upon* exiting the application. It also depends on the **Confirm** use case from the Warning feature existing as a prompt to save changes is a form of confirmation, as described by said feature.
- Finally, the **Long/UndoableUserAction** needs no other use case (within the scope of this meta-model) to be viable. As for the borrowed use cases, it needs SaveToHistory from the Undo feature and ShowProgress from the System Progress Feedback feature.

The remaining dependencies that exist among the borrowed use cases (those represented in italics in Table 5.3-2) as well as the corresponding rows are omitted to avoid redundancy. These relationships are explained in their respective sections: Section 5.5 for the ShowProgress use case, 5.2 for the Undo and SaveToHistory use cases and section 5.7 for the Confirm use case.

Table 5.3-2 Usability Use Case Dependencies: Abort Functional Usability Feature

	A_UC-1 Cancel Command	A_UC-2 Exit	A_UC-3 Save Changes	A_UC-4 Long/Und. UserAction	SPF_UC-1 Show Progress	U_UC-1 Undo	W-UC-3 Confirm	U_UC-6 STH
A_UC1 Cancel Command	-			X		X		
A_UC-2 Exit	X*	-	X*	X*		X*		
A_UC-3 Save Changes		X	-				X	
A_UC-4 Long/ Undoable User Action				-				X

From these use cases at least two minimal subsets can satisfactorily be implemented. A system including the Long/Undoable User Action along with CancelCommand is a potential configuration, as is one that only includes the Exit and SaveChanges use cases. Yet, while both these options are viable and may result from the elicitation process of a system, they are the exception rather than the norm, as most systems that contemplate cancelling commands (and handling their state) will likely require handling them at exit time as well.

#### 5.3.1.4 System Responsibilities for Usability

Table 5.3-3 presents the System Responsibilities for Usability for the present feature.

Table 5.3-3 System Responsibilities List for Abort

<b>System Responsibilities List for Abort</b>
A_SR-1 Identify cancellable commands The system is responsible for keeping track of commands that are cancellable
A_SR-2 Cancel commands and handle application state The system must allow users to cancel (cancellable) commands and to handle app state appropriately
A_SR-3 Exit application handling on-going commands/operations The system must provide access to Exit feature to close application while handling state
A_SR-4 Handle potential changes to be saved If changes exist at exit time, the system must allow the user to save them appropriately

These System Responsibilities for Usability are derived from the Usability Elicitation Clusters identified in section 5.3.1.2 as follows:

**A\_EC\_1 Identifying cancellable commands:** As mentioned earlier, this elicitation cluster contemplates knowing which system commands are cancellable. To do so, the system must keep a record of said commands, or, if they are too many or an expanding group of commands, the system must at least be aware of the type of command that shall always be cancellable. Thus, this elicitation cluster would yield the System Responsibility **A\_SR-1 Identify cancellable commands**.

**A\_EC\_2 Cancelling commands and handling application state:** This elicitation cluster covers how each command, or type of command, is to be cancelled and how the resulting system state must be handled. By debating the discussion items contained in it, stakeholders determine exactly what parts of the system’s state are affected and how they are to be restored after cancelation. It is thus the responsibility of the system to properly cancel these commands and undo any potential effects: **A\_SR-2 Cancel commands and handle application state**.

**A\_EC\_3 Exiting application while handling potential on-going commands/operations:** As determined in the discussion items in this elicitation cluster, the system must provide the user a way to exit the application at any given time and also to address any ongoing commands or operations when doing so. This yields the System Responsibility to **A\_SR-3 Exit the application handling on-going commands and operations**

**A\_EC\_4 Handling potential changes to be saved:** In systems where the concept of ‘saving changes’ is relevant (i.e. where user data files are changed by the use of the application) this elicitation cluster determines if and how said action will be prompted upon exiting the system. The system is responsible for detecting when to prompt the user to save changes and for how to do so, as described in the last System Responsibility: **A-SR-4 Handle potential changes to be saved**

Table 5.3-4 maps the relationships between the Usability Elicitation Clusters and the Usability System Responsibilities described above, for easy reference. Any project determined to require a specific Elicitation Cluster will also require its related System Responsibilities. Likewise, if a cluster is discarded during elicitation, its related responsibilities will not be a part of the resulting system. In the case of Abort this relationship is one-to-one.

Table 5.3-4 Usability Elicitation Clusters / System Responsibilities for Usability Mapping for Abort

Use Cases	Dependent Responsibilities
A_EC-1 Identifying Cancellable Commands	A_SR-1 Identify cancellable commands
A_EC-2 Cancelling commands and handling application state	A_SR-2 Cancel commands and handle application state
A_EC-3 Exiting application while handling potential on-going commands/ops	A_SR-3 Exit application handling potential on-going commands/ops
A_EC-4 Handling potential changes to be saved	A_SR-4 Handle potential changes to be saved

### 5.3.2 Usability Guideline for Software Development: Design artifacts

The Abort feature has a close relationship with the Undo feature in that cancelling a command or operation may entail undoing any potential partial results in which they may have incurred. As such, its design guideline will make use of the same concepts as the Undo feature regarding the ‘Command’ pattern by Gama et al [37] when defining said portion of its functionality.

In section 5.3.2.1 the System Responsibilities are brought to a lower abstraction level and describes them as High-level Design Component Responsibilities, and section 5.3.2.2 as

Low-level Design Component Responsibilities (for a MVC architecture). Finally, section 5.3.2.3 presents the Usability Design Meta-models for said Low-level Design Component Responsibilities as object-oriented class and sequence diagrams.

### 5.3.2.1 High-level Design Component Responsibilities

In order to support the System Responsibilities identified 5.2.1.4 at design level, the following sections describe the suggested High-level Design Components and their responsibilities for the Abort Functional Usability Feature, shown in Table 5.3-5.

As mentioned earlier, the parts of this feature that relate to having undo capabilities make use of the Command pattern by Gamma et al. Components (and, in the following sections, classes and objects) like the “Command” component, are taken directly from this widely known design pattern to fulfill the needs that arise from the corresponding System Responsibilities. Another pattern used across all the guidelines is the GoF Observer pattern, a defining part of the MVC architecture itself. The rest of the components conform the original contribution for this feature.

When represented graphically, as is the case of the Usability Design Meta-models in section 5.2.2.3, all components, objects and/or classes belonging to the GoF Command pattern or the Observer pattern are depicted in yellow and red, respectively (see page 71 for color legend of existing patterns).

#### 5.3.2.1.1 User Interface (UI) Component

This component is responsible for capturing all user input and forwarding any action invocations (possibly through a delegating component) to the part of the domain responsible for executing it. In this feature, calls to execute (and the abort execution of) an operation or command are captured by the UI Component. The UI Component is also responsible for relaying information to the user, like, for example, notifications to ‘save changes’ after attempting to close an application.

#### 5.3.2.1.2 Domain Component

A Domain Component represents the part of the system that is responsible for executing the actions requested by the user. As such, which class(es) ultimately implement the functionality herein described will vary from application to application. In a video program, for example, when choosing the ‘Export video’ option there will likely be a class (or group of classes) in charge of sending said video, perhaps after some form of conversion, to the hard drive as a file. Said class (or classes) would represent the Domain Component.

#### 5.3.2.1.3 Command Component

Based on Gamma’s definition of the Command Pattern, the Command Component is responsible for encapsulating method invocations and any pertinent state information at call-time.

As in the case of the Undo feature, when an cancellable (thus undoable) action is invoked through the UI, a new instance of Command is created. This instance is initialized with the signature of the method that is being called and a reference to the Domain Component in charge of executing it, for later invocation. It is also initialized with any state information that will need to be restored if the invoked action is ever ordered to be cancelled. Aside from storing this information, the Command Component is responsible for ultimately calling the invoked method in the Domain Command (once the state information has been saved), and for properly undoing said action (restoring the previous state information) should an ‘cancel’ order for it come from the UI Component.



#### 5.3.2.1.4 History Component

The History Component belongs to the Undo feature, and is used within Abort to provide undo capabilities when cancelling commands or operations. As in Undo, this component is responsible for storing all Command instances created within the system. Once an cancellable/undoable call is placed in the UI, the Command Component is created immediately stored in the History Component. If a call to cancel the latest action is placed in the UI, the History Component is responsible for locating (within itself) the Command instance which initially executed said action and ordering it to *undo*, to return the system to the state prior to execution, thus effectively cancelling it.

#### 5.3.2.1.5 Save Manager Component

The Save Manager Component is responsible for keeping track of any changes that may have occurred within the system that would potentially need to be saved. It is also responsible for alerting the application (and ultimately, the user) of the existence of such changes upon invocation of the 'exit' feature and of saving them to the appropriate media if requested.

Table 5.3-5: Usability Guideline: High-level Design Component Responsibilities. Abort

System Responsibility	High-level Design Component Responsibilities
A_SR-1 Identify and execute cancellable commands	<p>A software component, preferably that responsible for handling user events (<i>UI</i>), must know of all the commands that are cancellable. By being in charge of this responsibility, it will be able to display the necessary interface components to provide the user with the means to cancel said command.</p> <p>The <i>UI</i> is also responsible for listening for command invocations from the user.</p> <p>Execution of actions is always the responsibility of the pertinent Domain Component in the application</p> <p>The component in charge of delegating actions (if any) should determine whether the action is undoable or not, from a pre-established list.</p> <p>If the action to execute is undoable, it must first be encapsulated as an instance of a Command Component, together with any pertinent state information and the necessary actions needed to revert its effects.</p> <p>Such an instance is then stored in a History Component, responsible for keeping a single (ordered) collection of all executed undoable actions.</p> <p>After encapsulation, the Domain Component is then free to execute the invoked action</p>
A_SR-2 Cancel commands and handle application state	<p>The <i>UI</i> must listen for user calls to cancel ongoing commands</p> <p>The component in charge of delegating actions (if any) is responsible for knowing which thread is running the command being cancelled and to order it to stop, as well as ordering the <i>History Component</i> (see Undo) to undo any effects caused by said command, returning the application to its original state.</p> <p>The <i>History Component</i> must then retrieve the last* <i>Command</i>, without discarding it, and order it to 'undo' itself.</p> <p>The <i>Command</i>, in turn, executes the necessary actions, using the stored state information, to return the system to the state preceding its execution</p>
A_SR-3 Exit application handling potential on-going commands	<p>When exiting the application, any on-going commands must be dealt with.</p> <p>The <i>UI</i> must listen for calls to exit the application</p> <p>If there are no on-going commands or operations, the application will exit immediately</p> <p>If there are on-going commands and/or operations, but the <i>UI</i> does not need to prompt the user for the type of exit s/he'd like to make, the <i>UI</i> must order all commands and/or operations to be dealt with in one of three ways (through an invoking component, if any):</p> <p>A) All on-going commands and/or operations will be cancelled immediately in the same manner (re: state retrieval) in which they are cancelled by users</p> <p>B) All on-going commands and/or operations will be allowed to finish execution, in which case the <i>UI</i> will wait until the last one notifies it has finished</p> <p>C) All on-going commands and/or operations will be terminated immediately (disregarding state) and the application closed.</p> <p>It is the <i>UI</i>'s responsibility to know whether or not to prompt the user for an exit type. If no prompt is made, it is also the <i>UI</i>'s responsibility to be aware of which type of exit it needs to make (i.e. the way in which commands and operations will be dealt with upon exiting).</p>
A_SR-4 Handle potential changes to be saved	<p>When the <i>UI</i> receives a call to exit the application, the component in charge of delegating actions (if any) should first ask a <i>SaveComponent</i> (see below) if there exist any pending changes before exiting</p> <p>A <i>SaveComponent</i> is responsible for determining whether there are changes to be saved and to order such saves.</p> <p>If there are changes pending to be saved, the delegating component will inform the <i>UI</i>, which in turn should prompt the user to save said changes</p> <p>These changes will be saved by the <i>SaveComponent</i> if requested</p>

### 5.3.2.2 Low-level Design Component Responsibilities for MVC

When instantiating for an MVC architecture, the High-level Design Components described above can be translated into the following system objects.

As is the case with most features presented in this work, the UI Component is instantiated by the **View** object and takes over all of its responsibilities except for the delegation of actions to other objects. This responsibility falls on the **Controller** object(s) of the MVC architecture.

Likewise, the Command Component is defined in the **Command** interface and implemented by **ConcreteCommand** objects, as described in the GoF Command Pattern [37]. For every cancellable (undoable) command the implementation will have a distinct ConcreteCommand class (i.e. Export VideoCommand, OpenGarageDoorCommand, etc.). Whenever a command is called through the View, the corresponding ConcreteCommand object will be created, saved to history to preserve state information in case of cancelation, and ordered to execute.

The History Component of the Undo feature is represented by the **HistoryList** singleton class, and covers all the responsibilities described for the History Component.

The Domain Component is instantiated by the **DomainClass**. Depicted in a different color in Figure 5.2-3, this is not an actual class but is rather a placeholder to be substituted at design time by the actual class that performs the task that was requested by the user.

The Save Manager Component is represented by the SaveManager class, which covers all of the components responsibilities regarding keeping track and storing any changes needing to be saved upon exit time.

The Low-level Design Component Responsibilities mentioned above are described in Table 5.3-6 as well as the way in which they carry out the System Responsibilities defined in section 5.2.1.4. For each System Responsibility, the sequence of actions required by the different objects is also presented as a set of UML diagrams.

Table 5.3-6: Usability Guideline: Low-level Design Component Responsibilities (MVC). Abort.

System Responsibility	Objects					Fig
	View	Controller	ConcreteCommand	HistoryList	DomainClass	
A_SR-1 Identify and execute cancellable commands	1. The <i>View</i> must listen for calls to commands. It must be aware of which of these are cancellable and provide the appropriate GUI components to enable cancellation.					Figure 5.3-4
	1. The <i>View</i> must listen for invocation of actions, <code>doAction()</code> . Upon reception, it must notify the <i>Controller</i> of said action	2. The <i>Controller</i> must determine if the invoked action is cancellable. In such case it must call the <code>execute()</code> method of the corresponding <i>ConcreteCommand</i> object (otherwise invocation goes directly to the <i>DomainClass</i> ), keeping a record of the <code>thread_id</code> in which <code>doAction()</code> is being executed. 3. The <i>Controller</i> must then <code>clone()</code> said <i>ConcreteCommand</i> and <code>add()</code> it to the <i>HistoryList</i> .	4a. Upon call to its <code>execute()</code> method, the <i>ConcreteCommand</i> first stores the necessary state information in its local variables. It then calls the appropriate method in the corresponding <i>DomainClass</i> (what was originally invoked) (0)	4b. The <i>HistoryList</i> saves the cloned <i>ConcreteCommand</i> atop its collection (so it can later be available to <code>undo</code> )	5a. The <i>DomainClass</i> executes the appropriate method to carry out what was originally invoked by the user through the <i>View</i> .	
	1. The <i>View</i> must listen for invocation of <code>cancel()</code> for a given thread. Upon reception, it must order the <i>Controller</i> to terminate said thread.(0) 7. The <i>View</i> must discard any <i>ProgressIndicators</i> upon notification, and also update any Smart Menus or History Displays (See Undo and Progress Feedback features)	2. The <i>Controller</i> orders the thread in which <code>doAction()</code> is being executed and orders it to <code>stop()</code> . It then orders the <i>HistoryList</i> to undo the last action for the corresponding <i>DomainObject</i> o.	4. Upon call to its <code>undo()</code> method, the <i>ConcreteCommand</i> calls the necessary methods in <i>DomainClass</i> (with any needed state information, stored upon execution) to revert its effects.	3. The <i>HistoryList</i> determines the <i>ConcreteCommand</i> to <code>undo</code> and calls its <code>undo()</code> method.	5. The <i>DomainClass</i> executes the methods invoked by <i>ConcreteCommand</i> . 6. The thread in which the <i>DomainClass</i> resides will then notify this to any existing <i>ProgressIndicators</i> (see Progress Feedback feature).	Figure 5.3-5
	A_SR-3 Exit application handling potential on-going commands	1. The <i>View</i> must listen for calls to <code>exit()</code> the application and determine if the user must be prompted for the <code>type</code> of exit to make. 2a. If so, the <i>View</i> prompts the user, whom responds with one of three possibilities ('cancel all', 'wait to finish' or 'immediate exit') 2b. If not, the <i>View</i> must simply forward said call to the <i>Controller</i> , along with what it knows to be the appropriate exit <code>type</code> . 3. Upon <i>Controller</i> notification, the <i>View</i> will kill the GUI and exit.	3. The <i>Controller</i> then proceeds to handle commands and operations according to the exit <code>type</code> . The <i>Controller</i> will a) order all commands to be cancelled as described in, and notify the <i>View</i> b) wait until all ongoing commands/operations notify it they have finished and then notify the <i>View</i> , or, c) simply notify the <i>View</i> .			
A_SR-4 Handle potential changes to be saved	1. The <i>View</i> must listen for calls to <code>exit()</code> and forward the call to the <i>Controller</i> 3. If there are changes to be saved, the <i>View</i> prompts the user. Upon okaying, the <i>View</i> orders the <i>Controller</i> to <code>saveChangesAndExit()</code>	2. The <i>Controller</i> asks the <i>SaveManager</i> if there are <code>pendingChanges()</code> . If so, the <i>Controller</i> notifies the <i>View</i> . Otherwise, execution of continues. 4. The <i>Controller</i> , in turn, asks the <i>SaveManager</i> to <code>saveChanges()</code>				Figure 5.3-6

### 5.3.2.3 Usability Software Design Meta-models

These Usability Software Design Meta-models are the UML diagrams representing the Low-level Design Component Responsibilities described in earlier. The following sections describe the class diagram and the classes involved in this feature and their interrelationships, followed by the descriptions of the sequence diagrams.

#### 5.3.2.3.1 Class Diagram

Figure 5.3-3 below shows the class diagram for the Abort Functional Usability Feature. As described in the Low-level Design Component Responsibilities Table, the main objects involved are the View, Controller, SaveManager, HistoryList, ConcreteCommand and DomainClass. The first two, fulfilling their role within MVC, respectively capture and distribute the user calls to perform actions. The SaveManager keeps a flag for pending changes to be saved, the changes themselves and the method to save them to the appropriate media. The HistoryList from the Undo feature, controls the list of actions (and their corresponding system states) that have been executed for potential canceling (undoing).

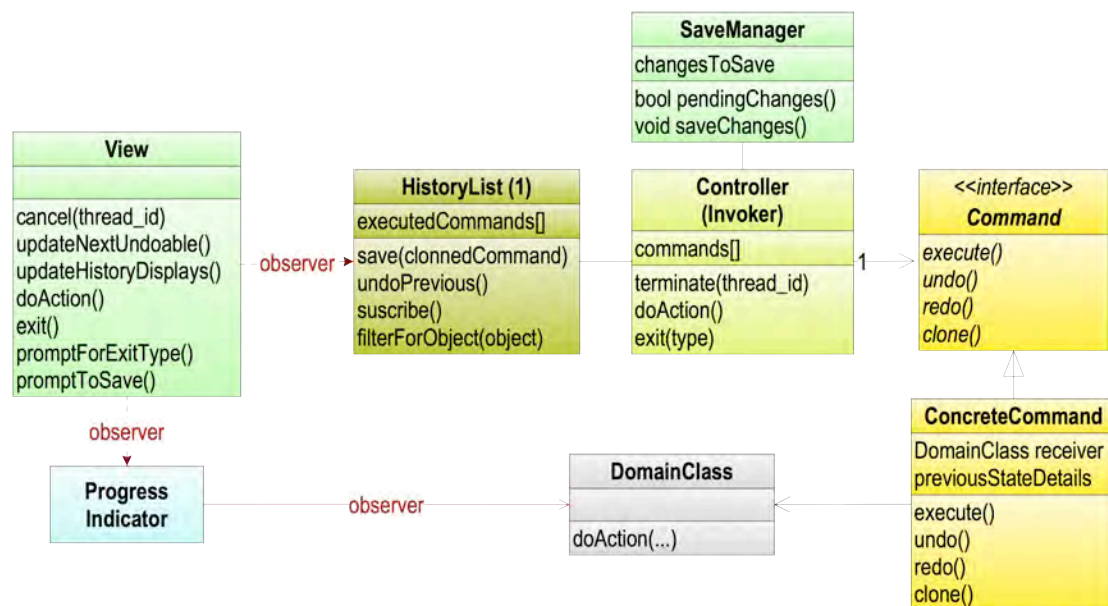


Figure 5.3-3: Usability design Meta-model. Class diagram. Abort.

As with other feature using the Command Pattern [38], the ConcreteCommand class implements a Command interface, as suggested by the authors, and is responsible for ordering the execution of the requested action (in DomainClass) as well as for storing all the necessary state information required for eventually cancelling it and reverting any partial effects it might have had. There must be as many ConcreteCommands as there are cancellable (and undoable) actions, and it is recommended they be labeled mnemonically. For example, the ConcreteCommand class in charge of invoking the exportVideo() method in a video application should be labeled something like ExportVideoCommand.

These classes and interfaces which belong to the Command Pattern of the GoF are depicted in their corresponding color, yellow to differentiate them from the rest of the classes particular to this meta-model.

HistoryList is depicted in yellow/dark green to show it belongs to the Undo pattern (dark green) but also implements some responsibilities proposed by Gamma's Command Pattern (yellow). Likewise, the Controller is depicted in yellow/light green, where light green is the Abort feature's color (the controller performs both original and Command Pattern functionality). The rest of the light green classes contain the Abort functionality, and the light blue, functionality from the System Progress Feedback feature.

### 5.3.2.3.2 Sequence Diagram "Executing Action"

Figure 5.3-4, shows the sequence diagram for executing actions described in Table 5.2-6. This diagram covers the invocation and execution of any cancellable action. As all cancellable actions are treated as actions that will potentially need to be undone, the sequence of events is exactly that of executing an *undoable* action, and thus this first diagram is shared with the Undo feature.

Following the GoF Command Pattern, prior to execution, the ConcreteCommand which encapsulates the invoked action (doAction()) is cloned (preserving its state information intact) and stored, in this case in the HistoryList object. After that, it is ordered to execute(), which entails a call to the desired method of the corresponding DomainClass.

When the invoked action is not undoable (second part of *alt* in Figure 5.2-4), the desired method is invoked directly off DomainClass and executed.

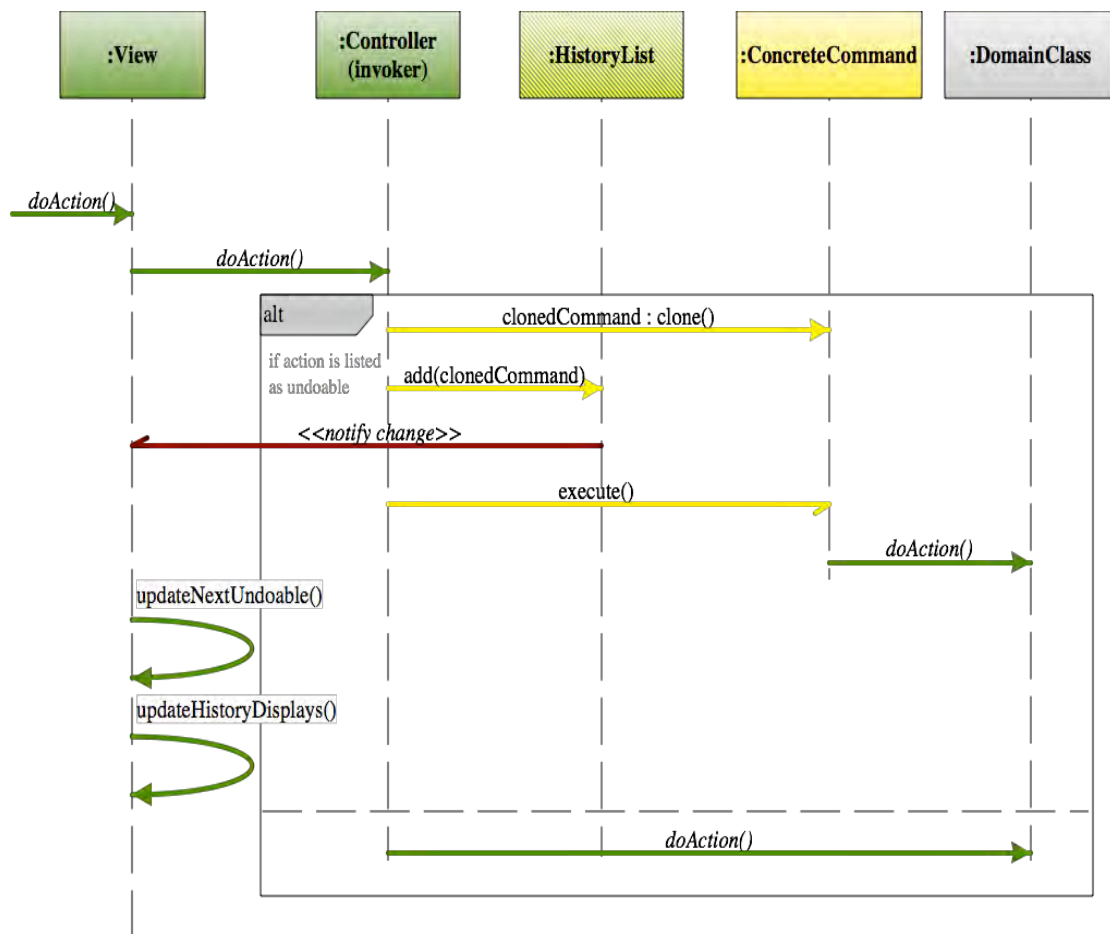


Figure 5.3-4: Sequence Diagram "Execute Action". Abort.

Classes and methods depicted in light green represent they belong to the Abort feature. However, this being a sequence adapted entirely from the Undo feature, due to the need for a cancellable action to be also undoable, all elements in this diagram belong either to Undo, or to existing patterns. Those in yellow are part of the Command Pattern and notifications in dark red represent Observer Pattern functionality [38]. The gray DomainClass is, as in all other features that contain it, a template class to be substituted at design time by the appropriate system class containing the actual cancellable action. For the full color legend see page 71

### 5.3.2.3.3 Sequence Diagram "Cancel Command"

Figure 5.3-5 describes the sequence for cancelling an on-going command.

This sequence starts when the user requests to cancel an on-going command. The View has the information that identifies said command and passes it onto the controller. With this information, the Controller finds the thread that the command is running in and orders it to stop. It then orders the HistoryList to undo whatever changes were produced by that command while it ran. The HistoryList orders the corresponding ConcreteCommand to undo, which leads to the DomainClass reverting the state to what it was before the start of the execution of the command.

Once the effects have been reverted the Controller orders the (stopped) thread to end (kill()). This sends a notification to any subscribed Progress indicators, which proceed to terminate as described in the Progress Feedback feature. Finally, the GUI updates the screen to reflect the command has been cancelled.

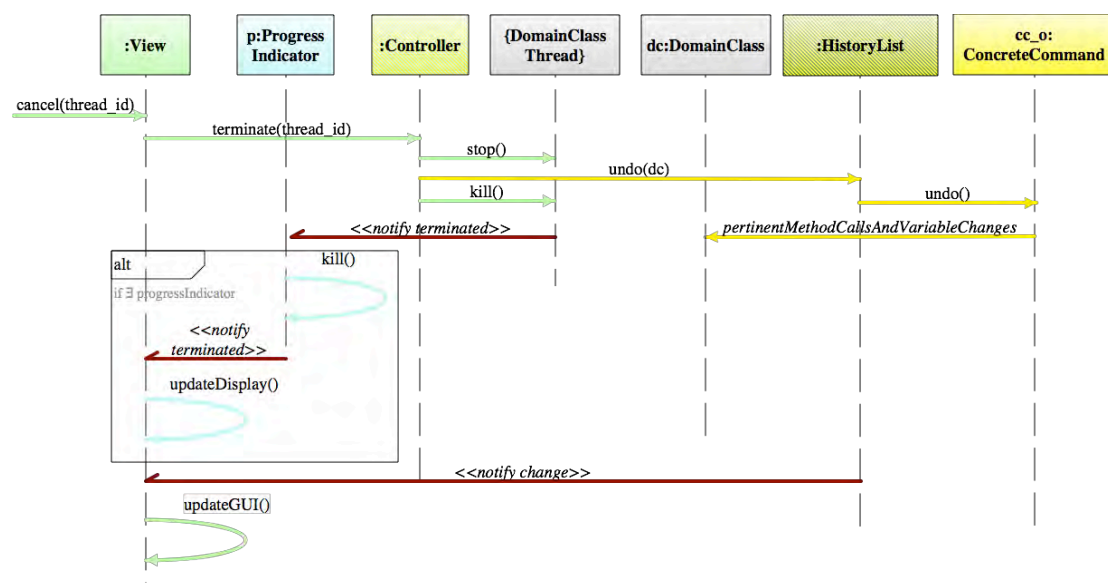


Figure 5.3-5: Sequence Diagram "Cancel Command". Abort.

### 5.3.2.3.4 Sequence Diagram "Exit application"

Figure 5.3-6 presents the sequence of exiting an application, with its different possible outcomes.

The sequence starts when the user requests to exit the current application. This call is forwarded to the controller, who asks the SaveManager if there are any changes pending to be saved. If there are, the Controller orders the View to ask the user to confirm whether or not he wants to save the changes. If changes are to be saved, the order is returned to the Controller who orders SaveManager to save the changes.

After changes have been dealt with, or when no changes are to be saved the View must determine the type of exit that will follow. It will only ask the user to make this decision in its stead if the system has been set up this way (determined at elicitation time).

Once the view knows the type of exit, it will forward this information to the controller, which can do one of two things, depending on this exit type: cancel all commands immediately (type 1) or wait for all of them to finish and then exit (type 2).

Whatever the exit type, once there's nothing left to cancel or save, the View terminates all graphic elements, ending the program itself.

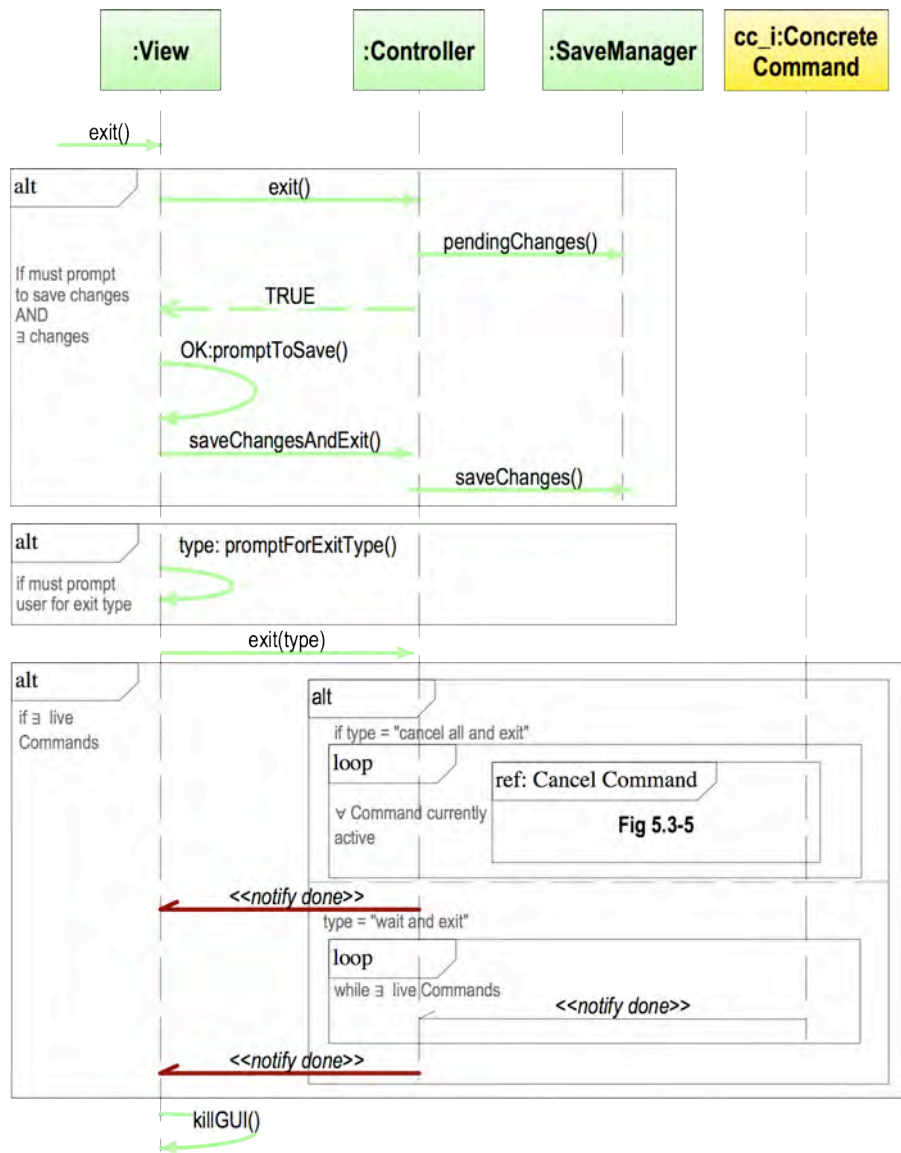


Figure 5.3-6: Sequence Diagram "Exit Application". Abort

## 5.4 Usability Guideline: 'Step-by-step'

The Step-by-step Functional Usability Feature covers allowing tasks with multiple steps to be represented as a series of navigable windows. Certain system tasks can require a series of inputs from the user that might not be feasible to perform within a single window or step. Also, more complex actions might entail decision-making on the part of the user during execution, making branching necessary.

The Usability Guideline for Software Development is made up of Analysis artifacts and Design artifacts. These are described for the present feature in the following two sections.

### 5.4.1 Usability Guideline for Software Development: Analysis artifacts

There are four artifacts to be used during the analysis phase: the Usability Elicitation Guideline, the Usability Elicitation Clusters, the Usability Use Case Meta-models, the System Responsibilities for Usability. These artifacts are described in the following four sections.

#### 5.4.1.1 Usability Elicitation Guideline

Table 5.4-1 shows the Usability Elicitation Guideline for the Step-by-step feature. This guideline contains three HCI recommendations, explained in the following sections.

##### 5.4.1.1.1 Step-by-step basics

For tasks that have (or that can be divided into) multiple steps, HCI authors recommend informing the user of the over-all goal to be achieved as of where he is in the process at every moment (SBS\_HCI-1). This is what is commonly known as the 'wizard-style' functionality involving a series of windows where information and decisions are requested from the user (SBS\_ELAB-1). The stakeholder discussions must cover which tasks are to be represented as wizards, and of which steps each is to be composed, as well as all the inputs and decisions required for these steps (SBS\_Q-1 to SBS\_Q-5). In Table 5.4-1, example SBS\_EX-1 "Payment Method" describes an example for this HCI recommendation.

##### 5.4.1.1.2 Step-by-step structure

As for the structure of wizards, HCI authors suggest that they can be linear or branch out in a tree-style fashion when user decisions are involved. In any case, users should always be informed of where they are in the sequence (SBS\_HCI-2). The use of 'breadcrumbs', or simplified map-like structures can be very useful in letting users know where they are within the navigation. (SBS\_ELAB-2). The stakeholder discussion for this HCI recommendation (SBS\_Q-6) contemplates if such breadcrumbs are to be used, how they will represent the navigational path, the current position of the user within it, past and future steps, and how they will handle overly large trees with multiple branches. Example SBS\_EX-2 "Checkout at Amazon.com" in Table 5.4-1 describes an example for this HCI recommendation.

##### 5.4.1.1.3 Navigation

Within the navigational sequence of a wizard, the user should be allowed to go back to a previous step, and to do so multiple times even until reaching the first step (SBS\_HCI-3). In doing so, the fact that actions may have been performed in previous steps, and the potential need to undo them when navigating backwards should be taken into account. The possibility of cancelling the wizard altogether at any time should also be considered. (SBS\_ELAB-3). Discussions SBS\_Q-7 to SBS\_Q-11 cover which tasks will provide 'back' and 'cancel' options, and what executing them will mean in terms of undoing previous actions. SBS\_EX-2 shows an example for this HCI recommendation in Table 5.4-1 "Movie Theater Reservations"



Table 5.4-1. Usability Requirements Elicitation Guideline. Step-by-step.

<b>Identification</b>			
Name	Step-by-step		
Family	Wizard		
Aliases	Wizard [49]; Step by step [42];		
<b>Intent</b>			
To allow tasks with multiple steps to be represented as a series of navigable windows			
<b>Problem</b>			
Certain system tasks require a series of inputs from the user that may not be feasible to perform in a single window. Also, more complex actions might entail decision-making on the part of the user during execution, making necessary.			
<b>Context</b>			
When a non-expert user needs to perform an infrequent complex task consisting of several subtasks where decisions need to be made in each subtask.			
<b>Interrelationships</b>			
The Step-by-step feature will need functionality from the Undo feature when dealing with wizards that incur in changes during navigation, as they would need to be undone when backtracking or exiting.			
<b>HCI Recommendation</b>		<b>Elaboration</b>	
<b>Discussions with Stakeholders</b>		<b>Usage Examples (optional)</b>	
SBS_HCI-1 Step-by-step basics	SBS_ELAB-1 User input	SBS_Q-1 Which tasks require the user to traverse multiple steps?	SBS_EX-1 Payment Method
Take the user through the entire task one step at a time [49] [42]. When the task is started, the user is informed about the goal that will be achieved and the fact that several decisions are needed. If information is needed from the user, ask for it in simple terms and with brevity; by keeping it short, you can better maintain the user's sense of flow through the whole step-by-step process [42].	The step-by-step functionality is most often seen in wizard-style applications  They often involve a series of windows or forms that require some kind of user input—i.e. yes or no decisions, multiple choice answers, filling in data fields—in each step of the process.	SBS_Q-2 For each of these, which steps are involved? SBS_Q-3 Which possible paths will the user take to navigate these steps? SBS_Q-4 For each step (of each task) what input will be required from the user? SBS_Q-5 Which of these inputs are required, which are optional, and for which can the system provide default values?	In applications where users have to choose a payment method at check-out time, the navigation will be split in as many 'branches' as payment methods exists, showing a different screen depending on the chosen method.  Mandatory fields (name, cc number) must be distinguished from optional fields.
SBS_HCI-2 Step-by-step structure	SBS_ELAB-2 Breadcrumbs	SBS_Q-6 How will tasks show the user info on where he is in the process?	SBS_EX-2 Check-out en Amazon.com
The task may branch like a flow chart, depending upon what information the user inputs, but the user doesn't necessarily need to know about all the available paths [49]. Users must be able to see where they are in the sequence and which steps are to be done, especially if there are more than 7 steps [42]. These groups may be thematic or alternatively, you may decide to split up based on decision points [42]. Note that the harder part is to balance the size and the number of the sub-sequences.	Breadcrumbs are often used to let the user know where s/he is in the process, particularly for longer sequences.  Keep in mind that only <i>linear</i> navigations can offer full bread crumbs (backwards and forwards). Trees with multiple navigation paths can, at best, display only the visited nodes.		In applications like the amazon.com cart, when the user makes a purchase he can see in the upper part of the screen the step of the multi-step process in which he is in at every moment (selection, cart, options, check-out, etc.)
SBS_HCI-3 Navigation	SBS_ELAB-3 Back and Cancel	SBS_Q-7 Which of these tasks will provide a "back" option to traverse steps in reverse order (or to skip back to an initial step)? SBS_Q-8 Which steps entail deep system commands to be performed? SBS_Q-9 For tasks that entail deep ops, will executing the "back" option undo such ops or simply allow the user to view the previous step (without undoing)? SBS_Q-10 Which of these tasks will provide a cancel option to exit the entire sequence? SBS_Q-11 Which type of cancel will be provided for each task?	SBS_EX-3 Movie Theater Reservation
If possible, allow the user to go back one step or to the beginning of the sequence. If there are more than 10 steps, try to break the task up into manageable sub-sequences, so it doesn't get too tedious for the user. The user must also be able to revise a decision by navigating back to a previous task [49], or back to the first step if needed [42]	The user must always be able to go back to a previous step. To provide this functionality correctly, the system will likely need to undo any outstanding operations performed in the later steps.  The possibility of leaving the wizard altogether should also be available in the form of a Cancel option.		In some applications destined to make movie theater reservations, once the user has chosen the movie (step 1) and the seat (step 2), if the user decides to go 'back' and chose a different movie, the reserved seats must be 'freed' in the cases when they may have been temporarily reserved for the user.

### 5.4.1.2 Usability Elicitation Clusters

The Usability Elicitation Guideline in Table 5.4-1 suggests eleven discussions items (SBS\_Q-1 to SBS\_Q-11) to be held with stakeholders in order to elicit all aspects of the Step-by-step Functional Usability Feature. These discussion items can be clearly divided into six initial groups, or clusters, as depicted in the Usability Elicitation Clusters in shown in Figure 5.4-1, relative to the portion of the Step-by-step functionality that they cover.

**SBS\_EC-1 Handling Tasks with Multiple Steps:** The discussion items in this cluster, shown in Figure 5.4-1, cover which system tasks will be comprised of multiple steps and what will these steps be for each.

**SBS\_EC-2 Handling Navigation Trees:** Once it has been determined which tasks will be comprised of multiple steps (wizards), stakeholders must discuss how they will be navigated to determine their structures (i.e. linear, tree, etc)

**SBS\_EC-3 Handling user input and preconditions:** The discussion items contained in this elicitation cluster deal with determining what information, if any, will be asked of the user in each step of every wizard, including any default values that may be provided by the system to help the user, information about input fields that may be optional, mandatory for moving forward, etc.

**SBS\_EC-4 Providing “breadcrumbs” information to user:** The sole discussion item pertaining to this subject determines how to show the users where in the navigational path they are at all times by using a ‘breadcrumbs’ structure.

**SBS\_EC-5 Providing “Back” functionality:** The discussion items in this cluster deal with the need for allowing the user to backtrack within the navigation and what effects this will have regarding any actions that may have been executed prior to doing so (i.e. undoing previous steps upon backtracking). Backtracking all the way to the initial step in the wizard directly (going ‘Home’) is also contemplated.

**SBS\_EC-6 Providing Cancel feature:** This elicitation cluster deals with the possibility of canceling a wizard altogether, how to do so, and what effects it would have on previously executed actions. The particular type of ‘cancel’ to implement can be any of those provided in the Abort feature and the user is referred to it for further details.

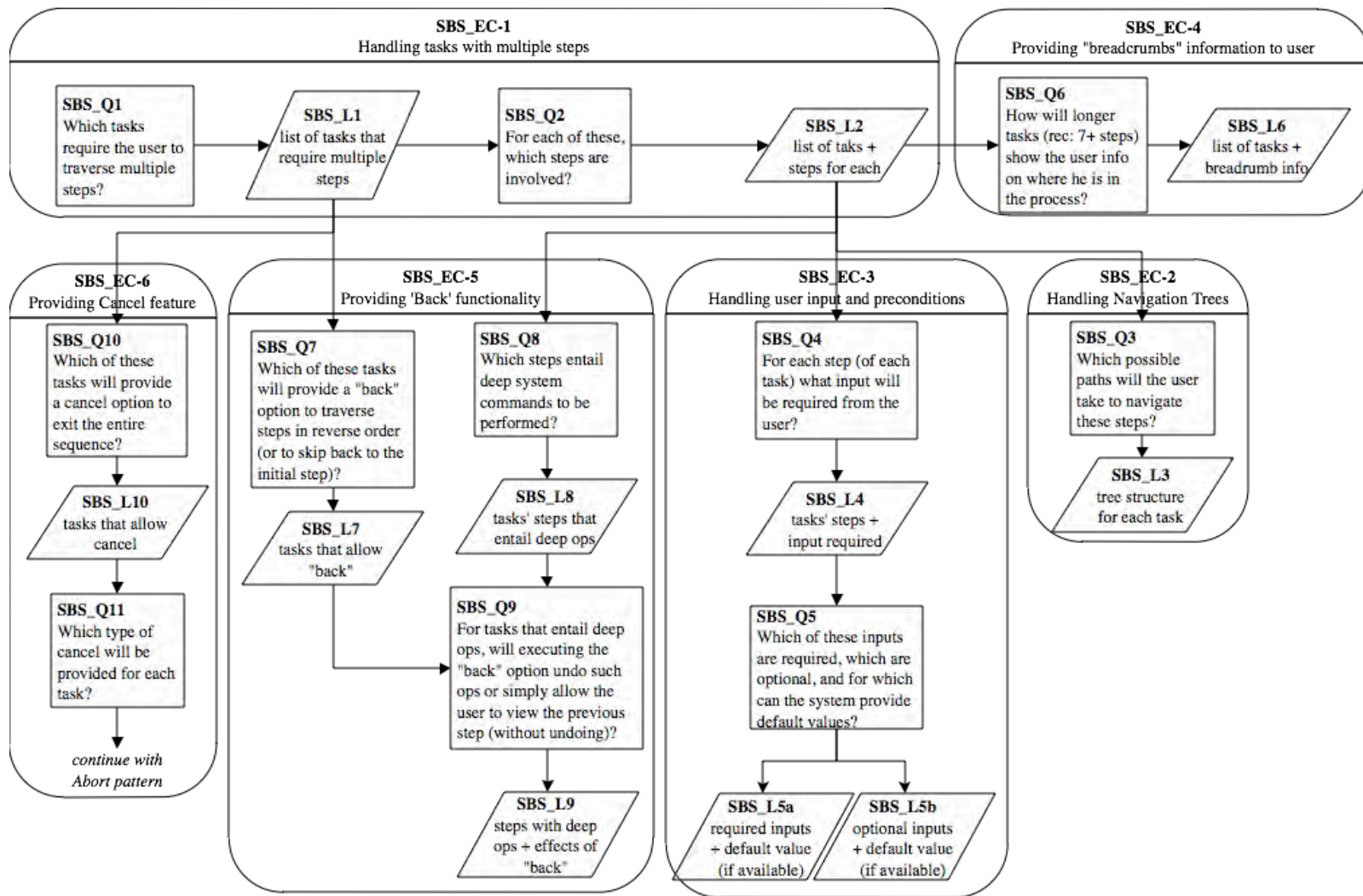


Figure 5.4-1: Elicitation Clusters. Step-by-step.

### 5.4.1.3 Use Case Meta-model

The Use Case Meta-model for the Step-by-step feature is shown in Figure 5.2-2 in which seven use cases are identified. Two are borrowed use cases from the Undo feature. Four are concrete use cases and one is a template use case, all belonging to the Step-by-step feature.

**SBS\_UC-1 LoadWizard:** the user requests the wizard to be loaded, which will present him with the first step in the wizard. Loading a wizard may or may not trigger a system action, represented by SBS\_UC-4 Undoable Action.

**SBS\_UC-2 NextStep:** By selecting the option ‘next’ at any time, the user requests the next step in the wizard. The system will present the user with this next step, and in doing so it may also trigger a system action, represented by SBS\_UC-5 UndoableAction. The next step is computed based on the structure of the sequence (linear, tree, etc) and any input given by the user in the current step.

**SBS\_UC-3 PreviousStep:** By selecting the option ‘back’, the user requests to go back to the previous step. The system will present the user with this previous step. In order to do so, it may need to undo actions that were executed while visiting it for the first time upon returning to it (see Undo). Regardless of the structure of the sequence being traversed, the previous step is always the last one that was visited prior to selecting the ‘back’ option.

**SBS\_UC-4 CancelWizard:** At any point during the navigation of a wizard the user requests the option to ‘cancel’ it, effectively exiting the sequence. Just as when going to the previous step, cancelling a wizard may incur in undoing any actions that were previously executed within it.

**SBS\_UC-5 UndoableAction:** This template use case represents any system action that may occur when going from one step of a wizard to another. For example, when clicking ‘next’ within an installation wizard, certain software components may be installed. These will need to be uninstalled if the user chooses to cancel the wizard or to traverse it backwards to, for example, make different choices, hence the need for these actions to be undoable actions and treated as such.

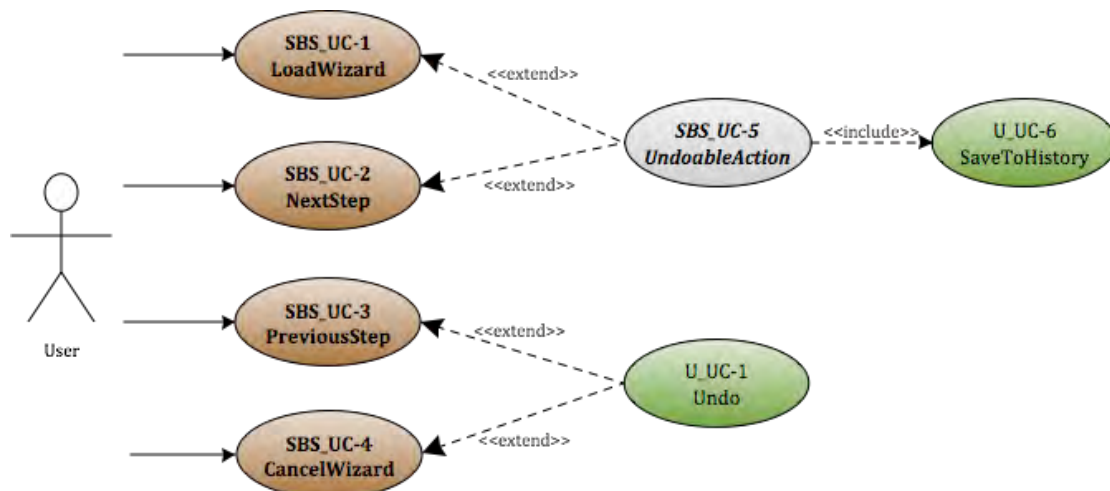


Figure 5.4-2 Use Case Meta-model. Step-by-step

As mentioned above, the applicability of each of these use cases will depend on the results of the elicitation process. If during elicitation of this feature it is determined that, for example, no intermediate steps will ever trigger any actions (i.e. all actions will only be carried at the end of the navigation) the borrowed use cases from the Undo feature (U\_UC-1 Undo and

U\_UC-6 SaveToHistory) will be discarded. Similarly, use cases depend on one another. These dependencies shown in Table 5.4-2 for Step-by-step.

- The **LoadWizard** use case has one conditional dependency (marked with an asterisk) with the template use case **UndoableUserAction**. This dependency occurs only when loading the wizard must trigger a particular system action, represented by the UndoableUserAction use case. Otherwise, the LoadWizard use case needs no other use case to be viable.
- The **NextStep** use case needs the **LoadWizard** use case to be viable, as a wizard must be loaded initially before the user should be allowed to navigate it. It also has the same conditional dependency as the LoadWizard use case with the **UndoableUserAction** in the instances when clicking ‘next’ must trigger a particular system action.
- The **PreviousStep** use case needs the **Undo** borrowed use case whenever clicking ‘back’ must undo a system action that was taken in the previous step. Furthermore, this use case is viable only when **NextStep** is also implemented, as in order to move ‘back’ the user must have first moved forward (‘next’).
- The **CancelWizard** use case needs the **Undo** borrowed use case in the same way PreviousStep does: in order to revert any actions previously taken within the wizard.
- Finally, the **UndoableUserAction** will always need **SaveToHistory**, as any action that is to be undoable must be saved upon execution (along with its state.).

The remaining dependencies that exist among the borrowed use cases (represented in italics in Table 5.2-2) are omitted herein to avoid redundancy (see Undo and Abort guidelines).

Table 5.4-2 Usability Use Case Dependencies: Step-by-step Functional Usability Feature

	SBS_UC-1 Load Wizard	SBS_UC-2 Next Step	SBS_UC-3 Prev Step	SBS_UC-4 Cancel Wizard	SBS_UC-5 Undoable UsrAction	U_UC-1 Undo	U_UC-6 Save to History
SBS_UC-1 LoadWizard	-				X*		
SBS_UC-2 NextStep	X	-			X*		
SBS_UC-3 PrevStep	X	X	-			X*	
SBS_UC-4 CancelWizard	X			-		X*	
SBS_UC-5 UndoableUsrAction					-		X

The relationships among the concrete use cases in this feature are relatively simple, though it’s worth noting that by reading vertically, it becomes evident that loading the wizard is needed for the rest of the use cases to make sense, as expected.

#### 5.4.1.4 System Responsibilities for Usability

Table 5.4-3 shows the proposed System Responsibilities for Usability for the present feature.

Table 5.4-3 System Responsibilities List for Step-by-step

System Responsibilities List for Step-by-step
SBS_SR-1 Load Wizard The system must know which actions are comprised of multiple steps and how to load them
SBS_SR-2 Go To Next The system must know what information to show when the user chooses to go to the ‘next’ step
SBS_SR-3 Handle user input and preconditions The system must update the GUI accordingly when at first and last nodes, and when preconditions haven’t been met to move to the next node
SBS_SR-4 Update breadcrumbs The system must keep breadcrumbs updated at all times to inform user place within navigation
SBS_SR-5 Go Back The system must allow the user to backtrack in the navigation, undoing the previous
SBS_SR-6 Go Home The system must allow the user to go to the first step in the navigation, undoing previous actions
SBS_SR-7 Cancel Wizard The system must allow the user to exit (cancel) the wizard, undoing any necessary actions

These System Responsibilities for Usability are derived from the Usability Elicitation Clusters identified in section 5.4.1.2 as follows:

**SBS\_EC-1 Handling tasks with multiple steps:** This Elicitation Cluster covers knowing which tasks will have multiple steps, and what these steps are. Therefore, the system should be responsible for knowing that a task that has been requested by the user is a wizard and for loading it appropriately. This gives way to the System Responsibility **SBS\_SR-1 LoadWizard**.

**SBS\_EC-2 Handling navigation trees:** To handle navigation of a wizard the system must be aware of its structure (tree-like for navigations involving user-decisions or linear for simpler processes). In doing so, every time the user selects to go to the ‘next’ step in the sequence the system must know the following: a) what the next step is (based on user input if a decision is involved), b) if any actions need to be executed as a result of moving forward in the sequence/tree. This gives way to the System Responsibility **SBS\_SR-2 GoToNext**.

**SBS\_EC-3 Handling user input and preconditions:** Whenever the user is at the first node, any option to go ‘back’ should be disabled in the GUI. The same happens in the case of moving forward (‘next’) when the user is at the last node. Similarly, when the user has not completed mandatory fields in the current step that are a precondition for moving forward, the possibility of moving forward should be disabled as well. This functionality is represented by the System Responsibility **SBS\_SR-3 Handle user input and preconditions**.

**SBS\_EC-4 Providing “breadcrumbs” information to user:** As mentioned earlier, this Elicitation Cluster entails providing the user information about where he is within the sequence at all times during navigation. This calls for the System Responsibility **SBS\_SR-4 Update Breadcrumbs**

**SBS\_EC-5 Providing “back” functionality:** The system is responsible for knowing which step was previously visited and for undoing any actions were executed within it when doing so. This functionality is represented in the System Responsibility **SBS\_SR-5 Go Back**. Furthermore, if the user selects to go back, not to the previous step but to the initial step in the sequence, the system should know which step that is and undo all actions taken since the start of the wizard. This is covered in the System Responsibility **SBS\_SR-6 Go Home**.

**SBS\_EC-6 Providing “cancel” feature:** Cancelling a wizard means leaving it altogether, destroying any interface elements associated with it, as well as with undoing any actions that the user may have executed while navigating it. The System Responsibility **SBS\_SR-7 Cancel Wizard** covers this functionality.

Table 5.4-4 maps the relationships between the Usability Elicitation Clusters and the Usability System Responsibilities for easy reference. Any project that requires a specific Elicitation Cluster will also require its related System Responsibilities. Likewise, if a cluster is discarded at elicitation time, its related responsibilities will not be a part of the system.

Table 5.4-4 Usability Elicitation Clusters / System Responsibilities for Usability Mapping for Step-by-step

Elicitation Clusters	Dependent Responsibilities
SBS_EC-1 Handling tasks with multiple steps	SBS_SR-1 Load Wizard
SBS_EC-2 Handling navigation trees	SBS_SR-2 Go To Next
SBS_EC-3 Handling user input and preconditions	SBS_SR-3 Handle user input and preconditions
SBS_EC-4 Providing “breadcrumbs” info to user	SBS_SR-4 Update breadcrumbs
SBS_EC-5 Providing “back” functionality	SBS_SR-5 Go Back SBS_SR-6 Go Home
SBS_EC-6 Providing “cancel” feature	SBS_SR-7 Cancel Wizard

## 5.4.2 Usability Guideline for Software Development: Design artifacts

The design artifacts of the Usability Guideline for Software Development for the Step-by-step feature are described in the following sections. The System Responsibilities described above are brought to a lower abstraction level as High-level Design Component Responsibilities in section 5.4.2.1. Section 5.4.2.2 expresses them as Low-level Design Component Responsibilities (for a MVC architecture). Finally, section 5.4.2.3 presents the Usability Design Meta-models for said Low-level Design Component Responsibilities as object-oriented class and sequence diagrams.

### 5.4.2.1 High-level Design Component Responsibilities

The following sections describe the suggested High-level Design Components and their responsibilities for the Step-by-step feature. As mentioned earlier, the parts of this usability feature that relate to having undo capabilities, make use of the Command pattern. Components like the “Command” component, are taken directly from this widely known design pattern to fulfill the needs that arise from the corresponding System Responsibilities. Another pattern used in these guidelines is the GoF Observer pattern, a defining part of MVC.

When represented graphically, as is the case of the Usability Design Meta-models in section 5.2.2.3, all components, objects and/or classes belonging to the GoF Command pattern or the Observer pattern are depicted in yellow and red, respectively (see page 71 for color legend of existing patterns). The rest of the components conform our original contribution for this usability feature.

#### 5.4.2.1.1 User Interface (UI) Component

This component is responsible for capturing all user input and forwarding any action invocations (possibly through a delegating component) to the part of the domain responsible for executing it. In this feature, calls to execute a wizard are captured by the UI Component, as well as those to cancel it and move forward/backward within it.

#### 5.4.2.1.2 Domain Component

A Domain Component represents the part of the system that is responsible for executing the actions requested by the user. As such, which class(es) ultimately implement the functionality herein described will vary from application to application.

#### 5.4.2.1.3 Command Component

Based on Gamma’s definition of the Command Pattern, the Command Component is responsible for encapsulating method invocations and state information at call-time. When an undoable action is invoked through the UI, a new Command is created and associated to the method that is being called and to the Domain Component in charge of executing it. Any state information that will need to be restored is also included in this Command.

#### 5.4.2.1.4 Tree Component

The Tree Component is responsible for knowing the entire structure of the sequence. It is also responsible for calculating what the next step in the sequence is at any time, based on the current step and any information the user may have entered in it. Furthermore, it must recall the previous step that was executed at any time, should the user select the option to go ‘back’ to it. Finally, it must be able to inform any interested parties (i.e. the UI Component) of where in the sequence the user might be at any given time.

#### 5.4.2.1.5 Node Component

A Node Component is responsible for handling all the information related to a single step in the wizard. It must know which information is to be displayed for and requested of the user and how to do so, as well as which system actions need to be invoked upon completion of said step of the sequence, if applicable.

Table 5.4-5: Usability Guideline: High-level Design Component Responsibilities. Step-by-step

System Responsibility	High-level Design Component Responsibilities
SBS_SR-1 Load Wizard	<p>The component responsible for handling user input (<i>UI Component</i>) must listen for calls to load a new wizard. It is responsible for knowing how to display the wizard on screen once called and where to display the wizard components, as well as any data within.</p> <p>Also under responsibility of the <i>UI</i> is the activation/deactivation of 'next' and 'back' buttons when appropriate (at first step, last step and when relating to fulfillment of mandatory fields)</p> <p>After a wizard has been loaded, the <i>UI</i> must now listen for calls from the user to move forward (next) or backwards (back) in the wizard. At either call, the <i>UI</i> will forward responsibility (as well as any data entered in the current step of the wizard) to a delegating component (if any), which will ask of the <i>Tree Component</i> to provide the next (or previous) node in the sequence.</p> <p>A <i>Tree Component</i> is responsible for knowing the structure of the wizard.</p>
SBS_SR-2 Go To Next	<p>Given a specific node in the tree, the <i>TreeComponent</i> must know how to calculate the next node to show to the user. To do so, it must also know the information provided by the user, which will determine the path within the tree through which the navigation will be performed. The simplest navigational style is that in which regardless of the user input there is only one (linear) path to follow, but more complex structures are supported.</p> <p>Whenever a node is reached, it is the responsibility of that individual node (represented by a <i>Node component</i>) in the <i>Tree</i>, to know if any further actions must be performed. The <i>Node</i> must order any (if at all) associated <i>ComandComponents</i> to execute the predetermined actions. These <i>ComandComponents</i> will in turn ask their <i>DomainClasses</i> to execute each particular action, and will be saved to <i>History</i> (See Undo).</p>
SBS_SR-3 Handle user input and preconditions	<p>A <i>Node Component</i> is responsible for managing all of the information pertaining to a particular step in a wizard. It must know the fields that compose it, whether or not a field is mandatory, default data (if any) for each field, textual descriptions for fields, etc. A <i>Node</i>, however, is unaware of its neighboring nodes and of how its node information might be displayed to the user, if at all (this is the <i>View's</i> responsibility)</p>
SBS_SR-4 Update breadcrumbs	<p>When a wizard is first loaded the <i>UI</i> should display all the 'breadcrumbs' that are possible. In a tree-like navigation wizard, it is only feasible to display the breadcrumbs for previous steps, as the steps that may follow any given node may not be known. Only linear wizards may show the entire sequence, highlighting the 'current' step.</p> <p>In tree-like navigation wizards, it is the responsibility of the <i>UI</i> to keep a record of the already-navigated steps, and to display them for the user, including the 'current' step, differentiated from the rest.</p> <p>In a linear wizard, the <i>WizardTree</i> can provide the <i>UI</i> with a list of the names of every node in the chain on load time.</p> <p>The <i>UI</i> would be responsible for keeping any displayed breadcrumbs up-to-date through navigation.</p>
SBS_SR-5 Go Back	<p>At any point in the sequence of a wizard, the user may chose to go back to the previous step (or to the first step in the sequence). When doing so, two possibilities arise: One is, if the current step did not involve command execution, going back to a previous step will merely involve displaying the <i>UI</i> components for that previous step. Similarly, when going back to a first state, if none of the executed steps entailed command execution, it will only be needed to display <i>UI</i> components for that first step.</p>
SBS_SR-6 Go Home	<p>However, if the current step did involve command execution, going back to the previous node will entail undoing such execution. Similarly when going back to first node</p> <p>Before ordering the display of <i>UI</i> components for the previous step, the delegating component (if any) must undo the last action associated with the current step in the wizard (see Undo) which was saved to <i>History</i> during forward navigation. Similarly when going back to a first step, when all actions associated with all executed steps must be undone</p>
SBS_SR-7 Cancel Wizard	<p>At any point in the wizard, the user must be allowed to cancel (exit). When doing so, any actions saved to <i>History</i> (if applicable) during execution must be undone by the same component regularly in charge of saving/undoing operations--the delegating component (if any)--and the <i>UI</i> components for the wizard discarded. (See Abort)</p>



#### 5.4.2.2 Low-level Design Component Responsibilities for MVC

When instantiating for an MVC architecture, the High-level Design Components described above can be translated into the following system objects.

As is the case with most usability features presented in this work, the UI Component is instantiated by the **View** object and takes over all of its responsibilities except for the delegation of actions to other objects. This responsibility falls on the **Controller** object(s) of the MVC architecture.

Likewise, the Command Component is defined in the **Command** interface and implemented by **ConcreteCommand** objects. For every undoable command the implementation will have a distinct ConcreteCommand class (i.e. Export VideoCommand, OpenGarageDoorCommand, etc.). Whenever a command is called through the View, the corresponding ConcreteCommand object will be created, saved to history to preserve state information in case of cancelation, and ordered to execute.

The Tree and Node Components described above are represented by the **WizardTree** and **WizardNode** classes, covering all of their responsibilities regarding the structure of the wizard and the contents of each of its steps, respectively.

The Domain Component is instantiated by the **DomainClass**. Depicted in gray in Figure 5.2-3, and as is the case in every other Functional Usability Feature where it appears, the DomainClass is not an actual class but is rather a placeholder to be substituted at design time by the actual class that performs the task that was requested by the user.

The Low-level Design Component Responsibilities mentioned above are described in Table 5.2-6 as well as the way in which they carry out the System Responsibilities defined in section 5.2.1.4. For each System Responsibility, the sequence of actions required by the different objects is also presented as a set of UML diagrams.

Table 5.4-6: Usability Guideline: Low-level Design Component Responsibilities (MVC). Step-by-step.

System Responsibility	Objects						Fig
	View	Controller	WizardTree	WizardNode	Concrete Command	Domain Class	
SBS_SR-1 Load Wizard	1. The View listens for user calls to <code>load()</code> a wizard. Upon receiving such a call, it forwards it to the Controller 5. The View receives the <code>firstNode</code> of the Wizard and displays it.	2. The Controller locates the appropriate WizardTree and orders it to <code>load()</code> 4. Controller passes on the <code>firstNode</code> to View (0)	3. The WizardTree will return the first WizardNode (and optionally all node names. See below)				Figure 5.4-4
SBS_SR-2 Go To Next	1. The View listens for user calls to the <code>next()</code> step of a wizard. Upon receiving such a call, it forwards it to the Controller, together with the information entered by the user in the current step	2. The Controller locates the appropriate WizardTree and orders it to load the <code>next()</code> Node, sending along the information provided by the user through the View	3. Using the user-entered information provided by Controller (if needed), the WizardTree determines the next WizardNode in the navigation and orders it to <code>setUp()</code> with the user-entered information.	4. When ordered to <code>setUp()</code> , the WizardNode processes the information and, if it is supposed to execute any actions it calls upon the corresponding ConcreteCommand to <code>execute()</code>	5. When ordered to <code>execute()</code> , the ConcreteCommand calls on its respective DomainClass to <code>doAction()</code>	6. Domain Class executes the called action.	Figure 5.4-5
SBS_SR-3 Handle user input and preconditions	1. Whenever the View receives a WizardNode to <code>display()</code> , it disables/enables any 'back'/next buttons, depending on the current node information. It also must highlight compulsory fields (information that is contained within the WizardNode) and not making a 'next' button available until the user has entered the required information.						Figure 5.4-5
SBS_SR-4 Update breadcrumbs	2a. If <code>load(wizard)</code> returns only the first Node of the wizard, the View will display it as the first element of the 'breadcrumbs' list. Every time a new Node is loaded, its name will be appended to said list and highlighted as 'current' 2b. If <code>load(wizard)</code> returns a list of all <code>nodeNames[]</code> in addition to the <code>firstNode</code> , the View will display all names in a 'breadcrumbs' list, highlighting the name of the <code>firstNode</code> . Every time a new Node is added to the list, its name will be highlighted as 'current'.		1. When a WizardTree is first called to <code>load()</code> , if it is a <code>linearTree</code> , it will return a list of all the node names, in addition to the first node in the sequence.				Figure 5.4-4 Figure 5.4-5
SBS_SR-5 Go Back	1. When a 'back' button is available, the View must listen for user clicks to said button and forward the event to the Controller 4. The View <code>displays()</code> the screen for <code>previousNode</code> appropriately.	2. The Controller then locates the appropriate WizardTree and orders it to <code>goBack()</code> to the previous WizardNode. 3. The Controller forwards <code>previousNode</code> to View	3. The WizardTree returns the <code>previousNode</code> to the Controller				Figure 5.4-6

System Responsibility	Objects					Fig	
	View	Controller	WizardTree	WizardNode	Concrete Command		Domain Class
SBS_SR-6 Go Home	<p>1. A "Home" button should always be available in the View, and if clicked, the View must forward the call to the Controller</p> <p>6. The View displays() the firstNode</p>	<p>2. Whenever a call to goBackToFirstNode() is received, the Controller orders to undo every action in the HistoryList related to the current wizard (more specifically to every WizardNode executed so far).</p> <p>3. Then, it orders the WizardTree to load the firstNode again</p> <p>5. Controller passes on the firstNode to the View</p>	<p>4. WizardTree loads the firstNode in the wizard and returns it to Controller</p>				Figure 5.4-7
SBS_SR-7 Cancel Wizard	<p>1. At any point during wizard execution, the user must be allowed to cancel(), exiting the wizard. The View will pass this order to the Controller.</p> <p>3. Once cancelled (see Abort feature) the wizard window must be discarded by the View.</p>	<p>2. Whenever a call to cancel() is received, the Controller must order to undo every action in the HistoryList related to the current wizard (more specifically to every WizardNode executed so far).</p>					Figure 5.4-8

### 5.4.2.3 Usability Software Design Meta-models

These UML diagrams represent the Low-level Design Component Responsibilities described in earlier. The following sections describe the class diagram and the classes involved in this feature and their interrelationships, followed by the descriptions of the sequence diagrams.

#### 5.4.2.3.1 Class Diagram

Figure 5.4-3 below shows the class diagram for the Step-by-step feature. The main objects involved are the View, Controller, WizardTree, WizardNode, ConcreteCommand and DomainClass. The first two, fulfilling their role within MVC, respectively capture and distribute the user calls to perform actions. The WizardTree controls the structure and navigation of the wizard, while WizardNode contains the data for each step in the sequence.

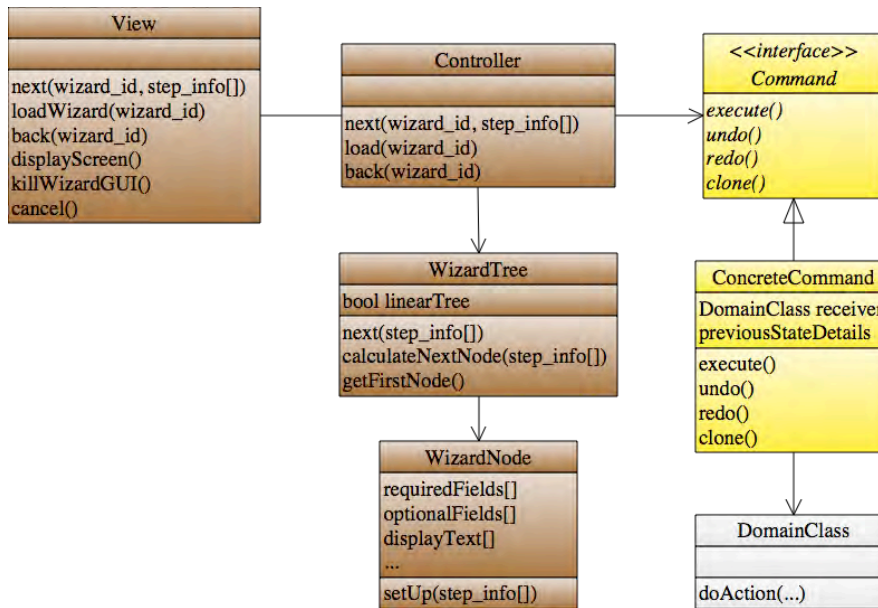


Figure 5.4-3: Usability design model. Class diagram. Step-by-step.

As described in previous sections where the ConcreteCommand class is referenced, it implements a Command interface responsible for ordering the execution of requested actions (in DomainClass) as well as for storing all state information required for eventually undoing the command it represents (i.e. the method it is calling within DomainClass)

Classes and methods in brown belong to the Step-by-step feature. Those in yellow are part of the Command Pattern and notifications in red represent Observer Pattern functionality. The gray DomainClass is the template class to be substituted at design time by the system class containing the undoable action. For the full color legend see page 71.

#### 5.4.2.3.2 Sequence Diagram "Load Wizard"

Figure 5.4-4 covers the process of initially loading a wizard and setting it up for use.

The user initially calls for the wizard to be loaded within the View, which prompts this class to forward the call to the Controller, who in turn determines the appropriate WizardTree class associated with the wizard being requested. The Controller then asks this WizardTree to load(), prompting it to return its first node object. Along with this node, the names for all the nodes in the wizard is also returned, in a structure of arrays mimicking the wizard structure (i.e. simple array for a linear structure, multi-dimensional array for trees) for future use by the View when displaying breadcrumbs. The Controller forwards the returned information to the View, which, in turn, updates the interface to show the first step of the wizard and the location of the user within it (i.e. in the first step).

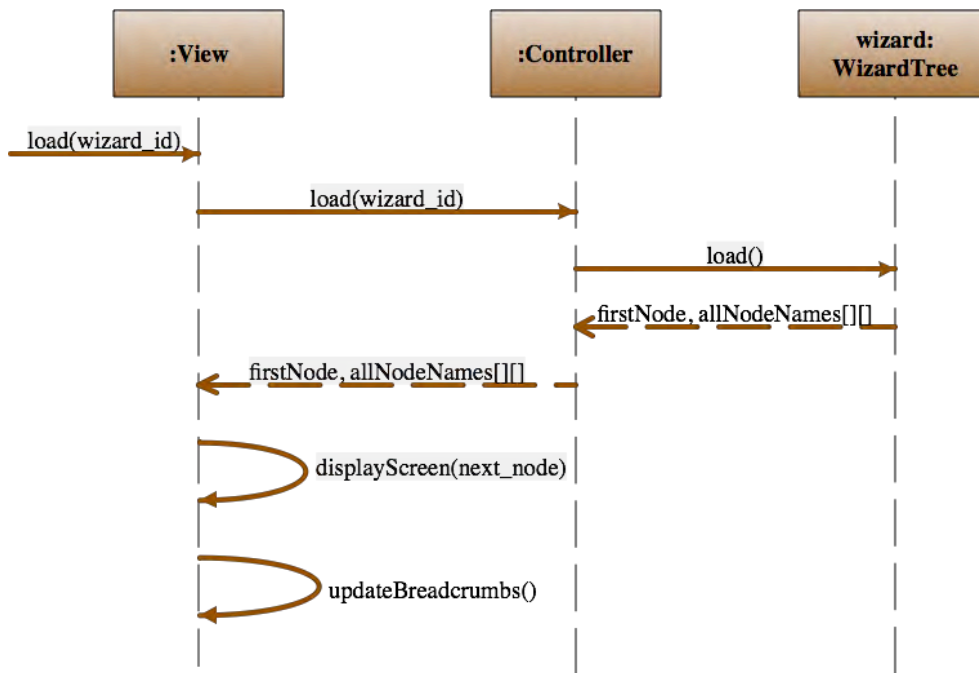


Figure 5.4-4: Sequence Diagram "Load Wizard". Step-by-step.

#### 5.4.2.3.3 Sequence Diagram "Next Step"

Figure 5.4-5 shows the sequence diagram for going from one step to the next in a wizard.

The sequence starts when the user selects the option to go 'next' from within any step of the wizard. This sends the View a reference to said wizard and any information that might have been entered by the user in the present step. This information is forwarded to the Controller which in turn finds the appropriate WizardTree for this wizard, which is the class in charge of calculating the next step in the sequence.

The WizardTree calls on the current node (`this_step`) to `execute()` any actions it may be assigned to before moving forward. If one exists, it is invoked from the corresponding ConcreteCommand as described by the Command Pattern and eventually executed off the corresponding DomainClass, after being saved to the HistoryList.

The WizardTree then calculates the next step to be shown to the user, based on any information provided in the current step (`step_info[]`) if applicable. This next node is then forwarded on to the Controller and finally the View, which displays it on screen while updating the bradcrumbs.

If no node is returned (`next_node` is null) it means the end of the wizard has been reached, which prompts the view to discard any GUI elements related to it, effectively ending it.

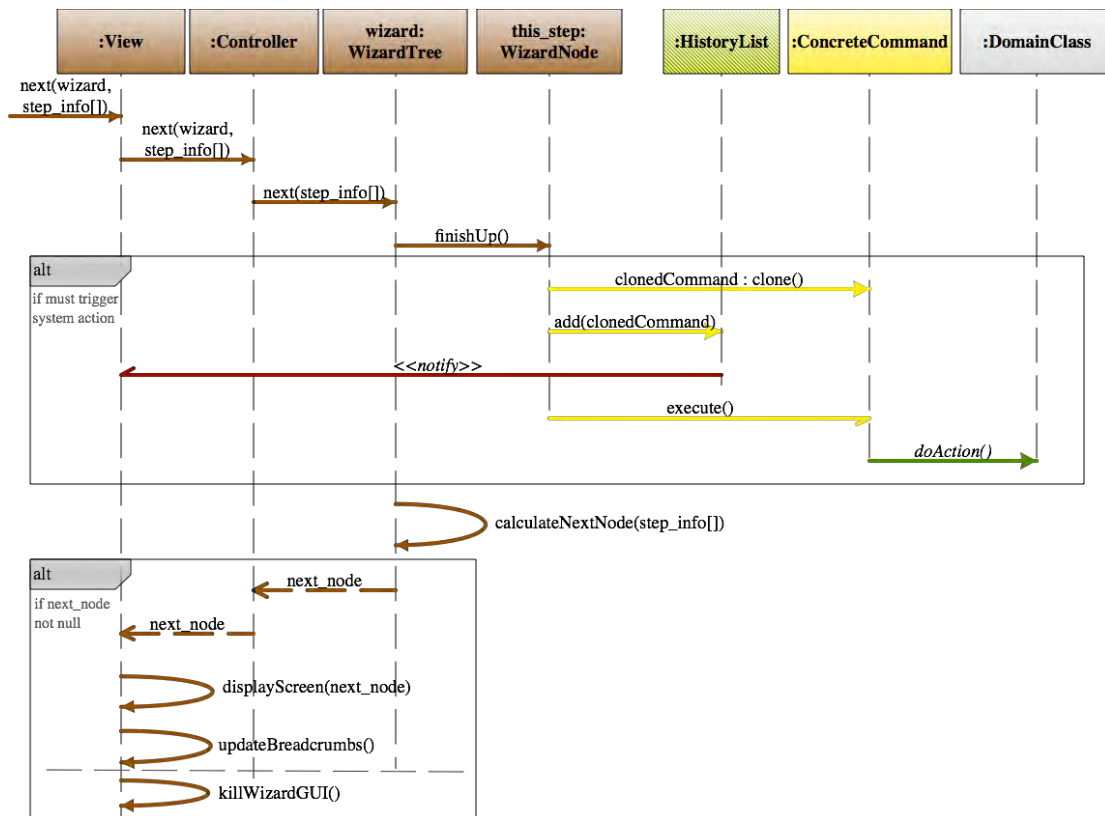


Figure 5.4-5: Sequence Diagram "Next Step". Step-by-step.

#### 5.4.2.3.4 Sequence Diagram "Go Back"

The sequence diagram for going back from one step to the wizard to the previous one is shown in Figure 5.4-6.

This sequence starts when the user selects the option to go 'back' one step, within any step of the wizard. This request is sent from the View to the Controller, which finds the appropriate WizardTree class that represents current wizard and requests it to go back(). In doing so, the WizardTree finds the Node object within it that represents the previous\_step in the sequence and orders it to revert() any action it may have triggered the last time it was visited. If such an action exists, the node will order it to undo as described in the Undo feature. After undoing, or in the cases when there is nothing to undo, the WizardTree will pass on the previous\_node onto the Controller, which will pass it on to the View. The View, finally, displays this node for the user, effectively making it the current step in the sequence, and updates the breadcrumbs accordingly.

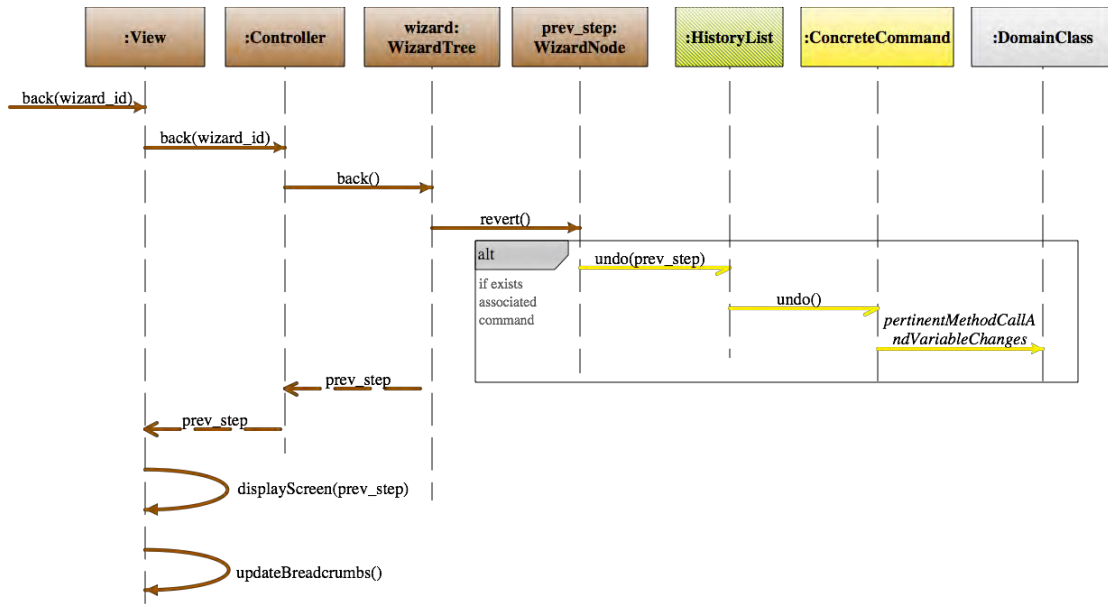


Figure 5.4-6: Sequence Diagram "Go Back". Step-by-step.

#### 5.4.2.3.5 Sequence Diagram "Go Home"

The sequence in Figure 5.4-7 shows the functionality of going "home": to the first step of the wizard.

Similarly to "Go Back", this sequence can be invoked from any step of the wizard by the user. The call is captured by the View, which forwards it to the Controller which in turn finds the appropriate WizardTree class that represents this wizard. The WizardTree locates every one of its nodes that have been traversed during the execution of the wizard and orders them to revert() any actions they may have triggered. As in the case of "Go Back" any triggered actions are undone, only in this case it's done multiple times ( i.e. for all steps).

Finally, the first\_node in the WizardTree is returned to the Controller and passed on to the View, which then displays it as the current node (effectively having sent the user "home" or to the first step of the wizard). Finally, the View rolls back the breadcrumbs to the first step.

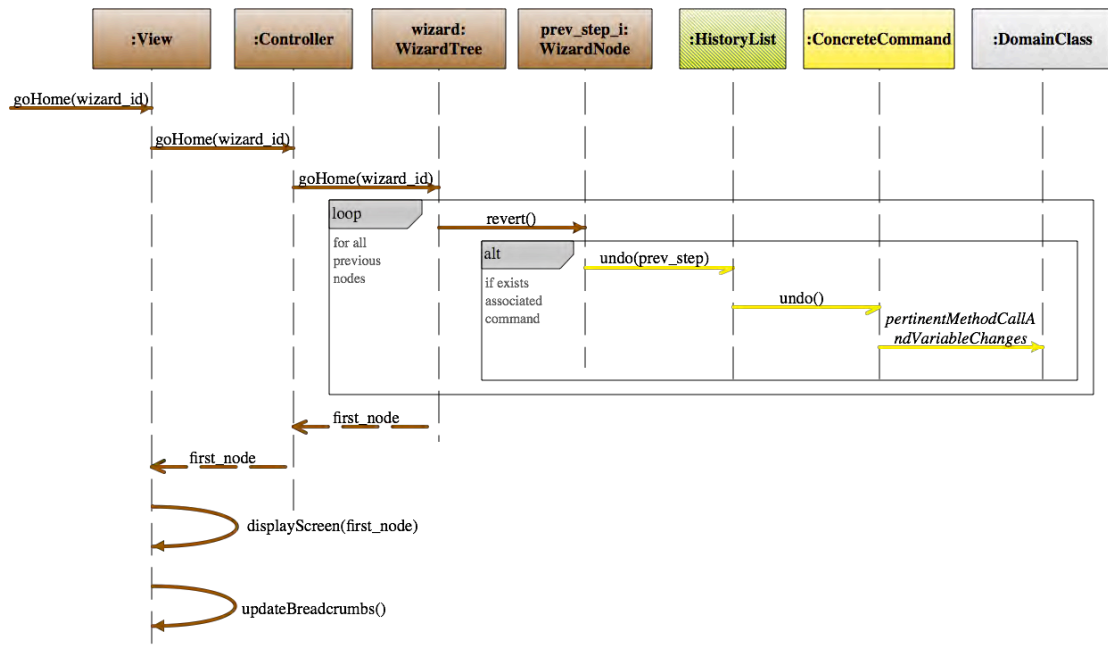


Figure 5.4-7: Sequence Diagram "Go Home". Step-by-step.

### 5.4.2.3.6 Sequence Diagram "Cancel Wizard"

Figure 5.4-8 shows the sequence for cancelling a wizard.

This sequence starts when the user selects the option to cancel the wizard at any step in the sequence. This request is sent from the View to the Controller, which finds the appropriate WizardTree class for the current wizard and requests it to cancel(). Much like when going "home" the WizardTree finds all the Node objects within it that have been traversed and orders them to revert() any action any of them may have triggered the last time they were visited. If such actions exist, each node will order them to undo as described in the Undo feature.

Once control returns to the View (after any and all actions have been undone), it discards all graphic elements related to the wizard, effectively terminating it.

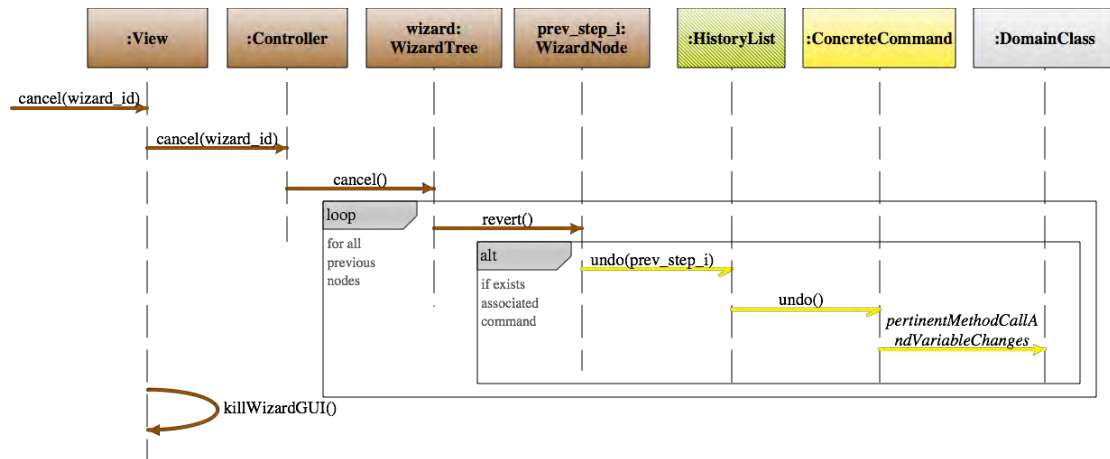


Figure 5.4-8: Sequence Diagram "Cancel Wizard". Step-by-step.



## 5.5 “Progress Feedback” Usability Guideline for Software Development

The Progress Feedback Functional Usability Feature covers the user’s need to provide the user with accurate visual feedback on the progress of the current task.

Certain system tasks will take a long time to execute and, therefore, the user needs to be informed of how much time remains in said tasks to s/he can make informed decisions in terms of whether to wait for the task to finish, cancel it, etc.

The Usability Guideline for Software Development is made up of Analysis and Design artifacts, described for the Progress Feedback feature in the following two sections.

### 5.5.1 Usability Guideline for Software Development: Analysis artifacts

There are four artifacts to be used during the analysis phase: the Usability Elicitation Guideline, the Usability Elicitation Clusters, the Usability Use Case Meta-models, the System Responsibilities for Usability. These artifacts are described in the following four sections.

#### 5.5.1.1 Usability Elicitation Guideline

Table 5.5-1 shows the Usability Elicitation Guideline (UEG) for the Progress Feedback Functional Usability Feature. In this guideline, there is a single HCI recommendation covering the basic characteristics of the progress feature, when to use it and what to show the user, as explained below.

##### 5.5.1.1.1 Progress Information Types

HCI authors suggest showing an animated indicator of how much progress has been achieved for an ongoing action if it is to take more than a few seconds to execute. This indicator should be as specific as possible (U\_HCI-1). Even when information is available to display a detailed progress indicator, the initial calculations can take a few seconds, during which an alternate (indeterminate) progress indicator should be displayed (U\_ELAB-1).

The stakeholders should discuss (see SBS\_Q-1 to SBS\_Q-11 for details) which system tasks are likely to need progress indicators (longer tasks), what information is available in each case for the calculation of the progress, and, lastly, how the system will handle cancellation of these tasks (see Abort)

In Table 5.5-1, SPF\_EX-1 “Feedback Examples” describes a variety of examples for this HCI recommendation.

Table 5.5-1 Usability Requirements Elicitation Guideline. Progress Feedback.

Identification			
Name	System Progress Feedback (SPF)		
Family	Feedback		
Aliases	Alias: Progress Indicator [42]; Progress [49]; Show Computer is Thinking, Time to Do Something Else [11]; Modeling Feedback Area [Coram, 96]		
Intent			
Provide the user with accurate visual feedback on the progress of the current task			
Problem			
Certain system tasks will take a long time to execute. The user needs to be informed of how much time remains in said tasks to s/he can make informed decisions in terms of whether to wait for the task to finish, cancel it, etc.			
Context			
When a time-consuming process interrupts the UI for longer than two seconds or so: [42]			
HCI Recommendation	Elaboration	Discussions with Stakeholders	Usage Examples (optional)
<p>SPF_HCI-1 Progress Information Types</p> <p>Show an animated indicator of how much progress has been made. Either verbally or graphically (or both). For tasks that take a long time (typically more than a few seconds) [49], tell the user:</p> <ul style="list-style-type: none"> <li>- What's currently going on,</li> <li>- What proportion of the operation is done so far,</li> <li>- How much time remains, and</li> <li>- How to stop it (or cancel it)</li> </ul> <p>About the remaining time: If the timing can be calculated, give an indication of the time remaining, either as a figure, or graphically, use either a Time-remaining Progress Indicator or a Proportion-completed Progress Indicator; if timing can not be estimated, but the process has identifiable phases, give an indication of the phases completed, and of the phases remaining. Use a Progress Checklist; if neither of these possibilities exist, then at least indicate the number of units processed (records, vectors ....); if no quantities are known – just that the process may take a while- then simply show some indicator that it's still going on, use an Indeterminate Progress Indicator.</p> <p>Verify that the application takes no longer than 1 second to display the progress indicator; and update the feedback at a rate that gives the user the impression that the operation is still being performed, e.g. every 2 seconds</p>	<p>SPF_ELAB-1 Indeterminate progress</p> <p>When a progress bar is first displayed the progress information might not be immediately available. If this is the case, and indeterminate progress indicator should be shown (in place of the determinate indicator) until accurate progress information can be calculated and displayed</p> <p>If progress cannot be refreshed every 2 seconds, an alternate (indeterminate) progress indicator should be visible to reassure the user that the task is still executing</p>	<p>SPF_Q-1 Which tasks are likely to take more than a few seconds (2 to 5) to complete, needing progress information to be displayed?</p> <p>SPF_Q-2 For which of these can actual progress be calculated?</p> <p>SPF_Q-3 For those whose progress can be calculated, which can provide the following information?</p> <p>SPF_Q-4 Identifiable phases completed</p> <p>SPF_Q-5 Time remaining for completion</p> <p>SPF_Q-6 Units processed</p> <p>SPF_Q-7 Percentage completed</p> <p>SPF_Q-8 For any remaining tasks (whose progress cannot be calculated) what kind of indeterminate progress indicator will be shown to the user?</p> <p>SPF_Q-9 For which tasks will a cancel option be provided to the user?</p> <p>SPF_Q-10 For the tasks listed above, how will the cancel option be provided?</p> <p>SPF_Q-11 For every task, what textual information (if any) will be shown to the user, together with the progress indicator?</p>	<p>SPF_EX-1: Feedback Examples</p> <p>As part of operating system behavior, progress bars are shown when copying large amounts of data within the hard drive or out to an external device.</p> <p>A cancel button (or the option to cancel through command-. for shorter processes) is also provided.</p> <p>The Mac OS progress bar will initially display a dialogue window with an indeterminate bar and a "calculating..." message. Once the remaining time is calculated it is displayed together with a determinate progress bar and the number of files remaining to be copied (SPF_3 parts 2, 3 and 4).</p> <p>Most software installers, particularly those that entail a long process like OS installers, provide the "phases completed" (SPF_3 1) information, together with multiple other forms of feedback.</p> <p>A checklist where completed phases are ticked off is most common when providing this type of feedback.</p>

### 5.5.1.2 Usability Elicitation Clusters

The Usability Elicitation Guideline in Table 5.5-1 suggests eleven discussions items to be held with stakeholders in order to elicit all aspects of the Progress Feedback Functional Usability Feature. These discussion items can be clearly divided into four initial groups, or clusters, as described in the Usability Elicitation Cluster Map, shown in Figure 5.5-1, according to the portion of the Progress Feedback functionality that they cover.

**SPF\_EC-1 Determining which tasks will require progress:** The discussion items in this cluster, as its name indicates, cover pinpointing those tasks which take more than a few seconds to execute and thus will require progress information to be shown.

**SPF\_EC-2 Calculating and Providing Progress Information:** This cluster covers what that information will be and how calculations will be performed for each type of progress display. For example, a Progress Bar display typically display the percentage completed/remaining, but can also show the number of units processed or remaining vs. the total number of units that need to be processed. In the case of the Progress Steps indicator, it is expected that the name of each step in the process be known and displayed as 'completed' before moving on to the next step.

**SPF\_EC-3 Providing Indeterminate Progress Information:** The discussion items in this cluster deal with the tasks for which progress must be shown, but little or no information is available about the task. In these cases, an indeterminate progress indicator must be displayed.

**SPF\_EC-4 Providing Textual Information:** This cluster deals with determining, for each task, which information will be displayed for the user in order to explain what the ongoing task is, and what the progress indicator means.

**SPF\_EC-5 Providing Cancel Option:** This last cluster deals with cancellation of an ongoing task for which progress is being shown. The user must refer to the Abort feature in order to determine how the system state will be handled.

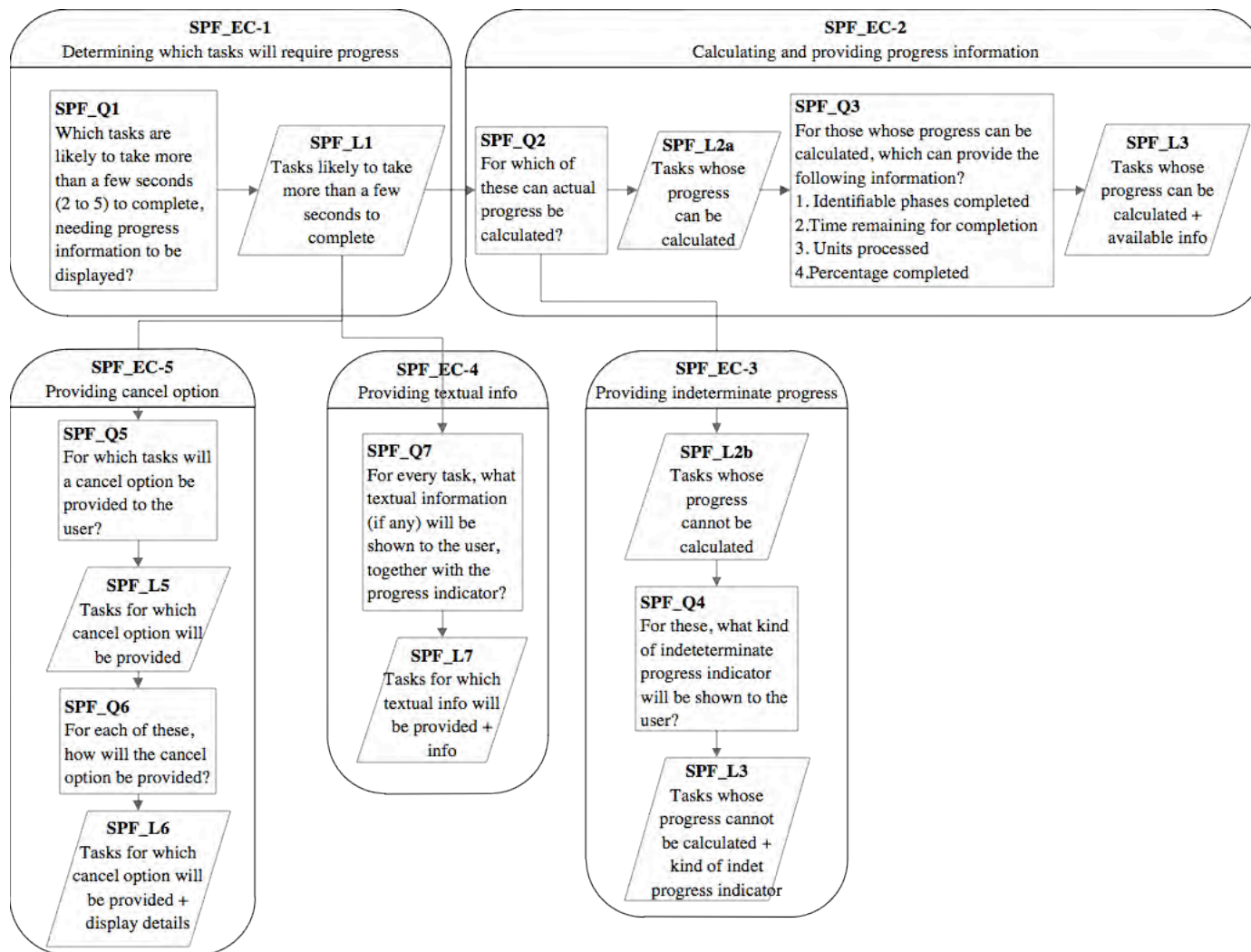


Figure 5.5-1: Elicitation Clusters. Progress Feedback

### 5.5.1.3 Use Case Meta-model

The Use Case Meta-model for the Progress feature is shown in Figure 5.2-2 (See page 71 for color legend), in which seven use cases are identified and described below.

**SPF\_UC-1 ShowProgress:** When the user requests a LongUserAction, this triggers the show of its progress throughout its execution. The ShowProgress use case can be: ShowProgressBar, ShowProgressSteps, ShowIndeterminateProgress, all of which are described below

**SPF\_UC-2 ShowProgressBar:** An empty bar is initially displayed, which continuously gets filled in as the task progresses. Textual information such as the percentage completed or the number of completed parts of the task may accompany this bar.

**SPF\_UC-3 ShowProgressSteps:** As the task requiring this type of progress feedback progresses, information about the 'steps' it is made up of is displayed. These actions typically have a set, finite number of steps, as opposed to those showing only progress bars.

**SPF\_UC-4 ShowIndeterminateProgress:** In this case, from the moment the action starts until it ends, a graphic element representing indeterminate progress is displayed (i.e. spinning wheel, clock, etc.). This use case may also be an extension of ShowProgressBar and ShowProgressSteps, at the very beginning of the task execution, before the initial progress information has been calculated in either case.

**SPF\_UC-5 LongUserAction:** This use case represents the system action that is directly invoked by the user, which takes several seconds to execute, thus needing progress information to be displayed for it.

**SPF\_UC-6 Cancel:** At any point in its execution, the user may cancel any LongUserAction, as described in the Abort feature.

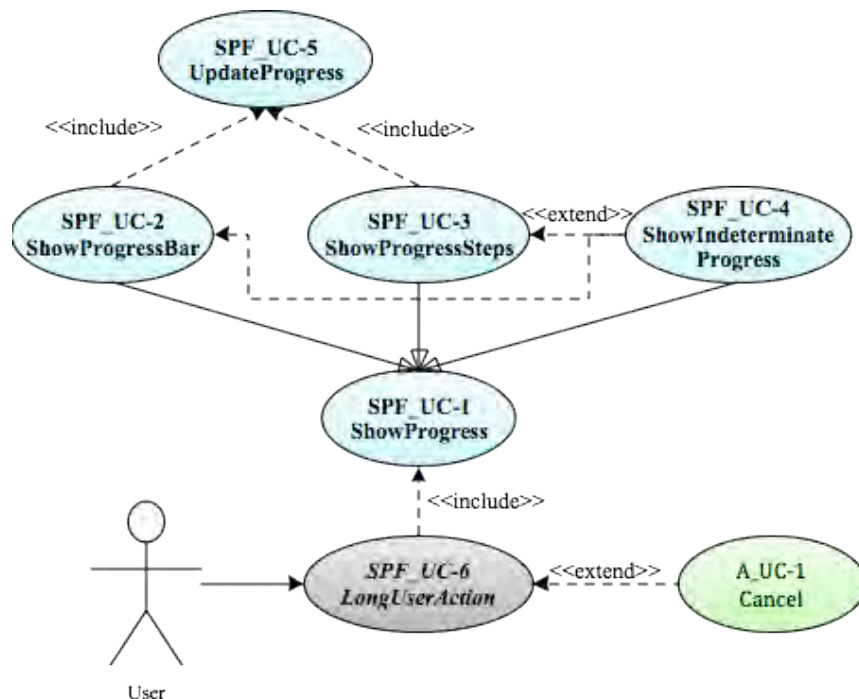


Figure 5.5-2 Use Case Meta-model. Progress Feedback

As mentioned earlier, the applicability of each of these use cases will depend on the results of the elicitation process. If during elicitation of the Progress feature, discussions determined that, for example, no task requires showing information about the ‘steps’ it has accomplished then the ShowProgressSteps use case would be discarded. Use cases also depend on one another. These dependencies are shown in Table 5.2-4, where we can see the following:

- The **ShowProgress** use case and its children evidently need the **LongUserAction** to exist in order to be viable. The children, in particular, need the parent use case. Furthermore, unless it contradicts what has been elicited for the Abort feature in a given project, all progress indicators should be cancellable.
- The **LongUserAction**, as it represents any long action in the system being developed, has no specific needs within this feature in order to be viable.
- Finally, in this context, the **Cancel** use case needs to be associated to an active show of progress in order to be viable (represented in the dependencies table by the parent use case **ShowProgress**).

Figure 5.5-3 Usability Use Case Dependencies: Progress Feedback

	SPF_UC-1 Show Progress	SPF_UC-2 Show ProgressBar	SPF_UC-3 Show Progress Steps	SPF_UC-4 Show IndetermProg	SPF_UC-5 LongUser Action	SPF_UC-6 Cancel
SPF_UC-1 ShowProgress	-				X	X
SPF_UC-2 ShowProgressBar	X	-			X	X
SPF_UC-3 ShowProgressSteps	X		-		X	X
SPF_UC-4 ShowInnetermProgress	X			-	X	X
SPF_UC-5 LongUserAction					-	
SPF_UC-6 Cancel	X					-

For such a small set of use cases, the only relevant conclusion that can be drawn from reading vertically is that, aside from the LongUserAction use case, the Cancel use case is pivotal to the Progress Feedback Functional Usability Feature, since, in most cases, any type of progress will require providing a way out for the user to terminate execution.

#### 5.5.1.4 System Responsibilities

Table 5.5-2 shows the proposed System Responsibilities for Usability for the present feature.

Table 5.5-2 System Responsibilities List for Progress Feedback

System Responsibilities List for Progress Feedback
SPF_SR-1 Determine which tasks will require progress The system must know which system actions might take long to execute
SPF_SR-2 Calculate and provide progress information The system must provide progress information for each action by using all available information
SPF_SR-3 Provide cancel option The system must allow users to cancel on-going actions
SPF_SR-4 Provide textual information The system must provide information about the task during progress display
SPF_SR-5 Provide indeterminate progress information The system must provide indeterminate progress information for tasks requiring it and when no other alternative is available

In section 5.5.1.2 we identified five elicitation clusters. Though in many other features a single cluster may yield more than one system responsibility or vice versa, in this case there is a one-to-one relationship between clusters and system responsibilities. Due to its simplicity in, the logic followed to derive each system responsibility is omitted in this chapter (each System Responsibility represents exactly the functionality covered by its corresponding cluster).

## 5.5.2 Usability Guideline for Software Development: Design artifacts

The design artifacts for the Progress feature are described below. In section 5.5.2.1 the System Responsibilities are brought to a lower abstraction level as High-level Design Component Responsibilities, and in section 5.5.2.2 as Low-level Design Component Responsibilities for MVC. Finally, section 5.2.2.3 shows the Usability Design Meta-models.

### 5.5.2.1 High-level Design Component Responsibilities

In order to support the System Responsibilities at design level, the following sections describe the suggested High-level Design Components for the Progress feature, shown in Table 5.6-5.

#### 5.5.2.1.1 User Interface (UI) Component

This component is responsible for capturing all user invocations and forwarding them (possibly through a delegating component) to the appropriate part of the domain, usually that responsible for executing the invoked action. Specifically for this usability feature, calls to execute an action that may trigger a progress display are to be listened for by the UI.

#### 5.5.2.1.2 Progress Component

As a part of the UI, the Progress Component is responsible for gathering the raw progress information from the Domain Component, processing it and displaying it for the user. For example, when copying files from one folder to another, the Progress Component is responsible for asking the Domain Component how many files are to be copied, and then for repeatedly requesting information on how many have already been processed. Internally the Progress component calculates a simple percentage and shows it to the user, alongside the number of files that have been copied, or those that remain. This way, the Progress Component not only relays the progress information on the interface, but also may previously perform additional calculations in order to produce the data to be shown.

#### 5.5.2.1.3 Monitoring Component

The monitoring component simply determines when a certain amount of time has elapsed, triggering the show of the progress display. For example, this time window may be set to two seconds, so any user-invoked task taking longer than two seconds to complete will trigger its corresponding progress indicator.

#### 5.5.2.1.4 Domain Component

A Domain Component represents the part of the system that is ultimately responsible for executing the (long) actions requested by the user. It is also responsible for periodically notifying all components that may be interested (in this case the Progress Component) of its current state of execution (i.e. processing unit 54 of 100).

Table 5.5-3: Usability Guideline: High-level Design Component Responsibilities. Progress Feedback feature.

System Responsibility	High-level Design Component Responsibilities
SPF_SR-1 Determine which tasks will require progress inf	The <i>UI Component</i> is responsible for knowing (from a pre-established list) whether an invoked action is among those that could potentially be 'long' (>2s)
SPF_SR-2 Calculate and provide progress information	The <i>UI Component</i> is responsible for listening for calls to these actions and for ordering their execution. If the action is among the potentially 'long', the <i>UI</i> must call unto an alternate <i>Monitoring Component</i> (preferably residing in a different thread) to determine when the allowed time (2s) has elapsed. When/if it does, the <i>UI</i> must start to display progress information (through a separate <i>Progress Component</i> , if needed) The component in charge of delegating actions (if any) is responsible for determining the <i>Domain Component</i> responsible for executing the invoked action and ordering it to do so. The <i>Domain Component</i> executing the action is responsible for continually notifying interested parties (namely the <i>UI</i> and/or <i>Progress Components</i> ) of the progress achieved and, eventually, its end. The <i>UI</i> and/or <i>Progress Components</i> are responsible for keeping the user up to date on the progress, based on notifications from the <i>Domain Component</i> .
SPF_SR-3 Provide cancel option	The component responsible for displaying the progress (be it the <i>UI</i> or an alternate <i>Progress Component</i> ) must provide a cancel option for the actions it knows to require one
SPF_SR-4 Provide textual information	The component responsible for displaying progress must also know of and display any needed textual information along with the progress details.
SPF_SR-5 Provide indeterminate prog info	When the <i>UI</i> component (or alternate <i>Progress Component</i> ) first displays the progress, it must do so indeterminately until it receives the first real progress update from the <i>Domain Component</i> .

### 5.5.2.2 Low-level Design Component Responsibilities for MVC

When instantiating for a MVC architecture, the UI Component is instantiated by **View** the object(s) and takes over all of its responsibilities except for the delegation of actions to other objects. This responsibility falls on the **Controller** object(s) of the MVC architecture.

The Progress Component is represented by the **ProgressIndicator**, and covers all the responsibilities described for the Progress Component in the previous section.

The **ProgressIndicator** class is meant to be extensible, and three proposed extensions are: the **ProgressBar**, meant for displaying information in percentage and n-of-m form; the **IndeterminateProgressIndicator** intended mainly for displaying progress feedback when no information is available about the on-going task; and, finally, the **ProgressSteps**, meant to display the steps that conform a task and to mark each as 'completed' when appropriate, for tasks where such information is relevant and available.

Finally, the Monitoring Component is represented by the **Monitor** object, which covers all of its intended functionality.

Table 5.5-4 details the Low-level Design Component Responsibilities described above and how they carry out each of the System Responsibilities defined for this feature. For each System Responsibility, the sequence of actions required by the different objects is presented as well as a set of UML diagrams that depict each of these interactions.

Table 5.5-4: Usability Guideline: Low-level Design Component Responsibilities (MVC). Progress Feedback.

System Responsibility	Objects					Fig
	View	ProgressIndicator	Monitor	Controller	DomainClass	
SPF_SR-1 Determine which tasks will require progress inf	1. The View must listen for invocation of actions and must determine (from a preexisting list) if the action being called could be potentially long.					
SPF_SR-2 Calculate and provide progress information	1. Notify the Controller when a long action has been invoked. 2. Ask the Monitor class to wait a specified amount of time (2s) 5. If the view is still waiting for the invoked action to return after Monitor responds, it starts up a ProgressIndicator (thread). 9. Whenever the ProgressIndicator notifies the View of new progress, the View will update the GUI 12. When the ProgressIndicator notifies the View that the progress has ended, it updates the GUI accordingly (no longer displaying the indicator)	6. The ProgressIndicator subscribes to the corresponding DomainClass for progress information. 8. Upon reception of progress information, the ProgressIndicator updates the progress, performing any needed calculations to do so. It notifies the View. 11. When ProgressIndicator receives a notification that the action being executed has ended it sets the progress to completed and notifies the View.	3a. The Monitor class starts up a clock and notifies the view after the time (2s) has elapsed.	3b. The Controller invokes the action, calling the appropriate class.	4. The DomainClass starts executing the invoked action 7. The DomainClass notifies its subscribers (namely the ProgressIndicator) of its progress 10. When the action is completed, DomainClass sends out a notification	
SPF_SR-3 Provide cancel option	1. When the View creates the ProgressIndicator it does so indicating via a boolean parameter whether the indicator will be cancellable or not	2. Depending on the parameter passed, the ProgressIndicator will enable a 'cancel' button or not.				
SPF_SR-4 Provide textual information	1. When the View creates the ProgressIndicator it must pass it the textual information to display	2. The ProgressIndicator holds this text and displays it alongside the progress information				
SPF_SR-5 Provide indeterminate progress information	1. Whenever a ProgressIndicator (that is not undetermined) is created, the view will initially paint it as indeterminate until the first progress update is received.					



### 5.5.2.3 Usability Software Design Meta-models

This section describes the UML diagrams representing the Low-level Design Component Responsibilities described above. Below, the class diagram is described, as well as the classes involved in this feature and their interrelationships, followed by the descriptions of the sequence diagram.

#### 5.5.2.3.1 Class Diagram

Table 5.5-4 below shows the class diagram for the Progress Feedback feature. The main objects involved are the View, Controller, ProgressIndicator, Monitor, ProgressBar, IndeterminateProgressIndicator, ProgressSteps and DomainClass. The first two, fulfilling their role within MVC, respectively capture and distribute the user calls to perform actions.

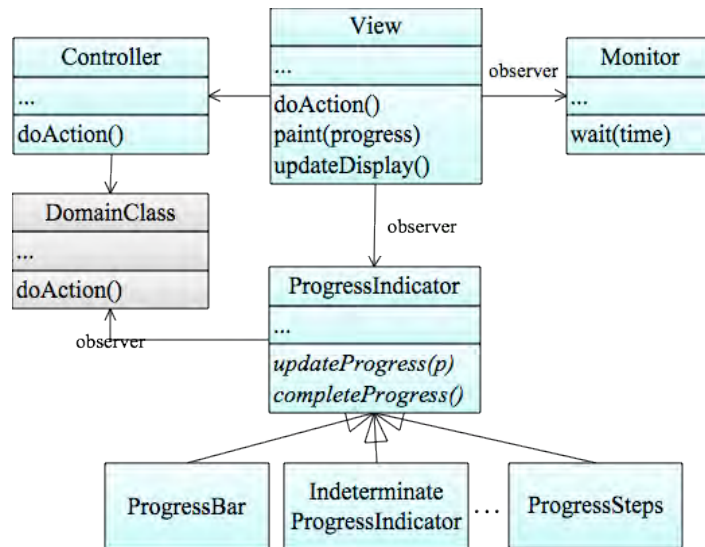


Figure 5.5-4: Usability Design Meta-model. Class diagram. Progress Feedback.

The ProgressIndicator, or more specifically, any of its subclasses, display the progress information for the task being executed by DomainClass. To do so, it subscribes to it and listens for the available information, processes it accordingly and displays it periodically.

The Monitor class simply starts counting milliseconds when requested to do so by the View and raises a flag when a pre-set time (usually 2 seconds) has been reached. With this information, the View can decide whether or not to call for the display of progress information for that particular action.

#### 5.5.2.3.2 Sequence Diagram “Show Progress”

Figure 5.2-4 shows the one sequence diagram for the Progress Feedback feature. It covers all Low-level Design Component Responsibilities previously described except for those related to the Abort feature (cancelling the task) as those are covered in section 5.3. This diagram covers execution of a long action and the display of its progress.

The sequence starts when the (template) action ‘doAction()’ is invoked by the user. The View captures this call and immediately orders the Monitor to start counting time, while simultaneously ordering the action to be executed through the Controller.

When the Controller gets the order to ‘doAction()’, it finds the appropriate DomainClass and orders its execution. If this execution finishes before the Monitor class reaches its pre-set time, then the sequence ends and no progress information is shown. If, on the other hand, the Monitor does reach its pre-set time before execution of the action ends, it notifies the View which proceeds to create a ProgressIndicator instance, appropriate for the action that was invoked.

This newly created ProgressIndicator subscribes to the DomainClass that is executing the action, expecting notifications of progress (i.e. units processed). While the DomainClass continues to send notifications other than that which indicates that the process has finished, the ProgressIndicator continues to update the progress information (including performing any needed calculations) and to forward this information to the View.

The View continuously updates the display until it receives the last notification (i.e. 100%) which causes it to update the display differently, by destroying all components related to the progress indicator.

This last notification is sent by the ProgressIndicator only after it receives a notification itself from the DomainClass, informing it that the process has ended.

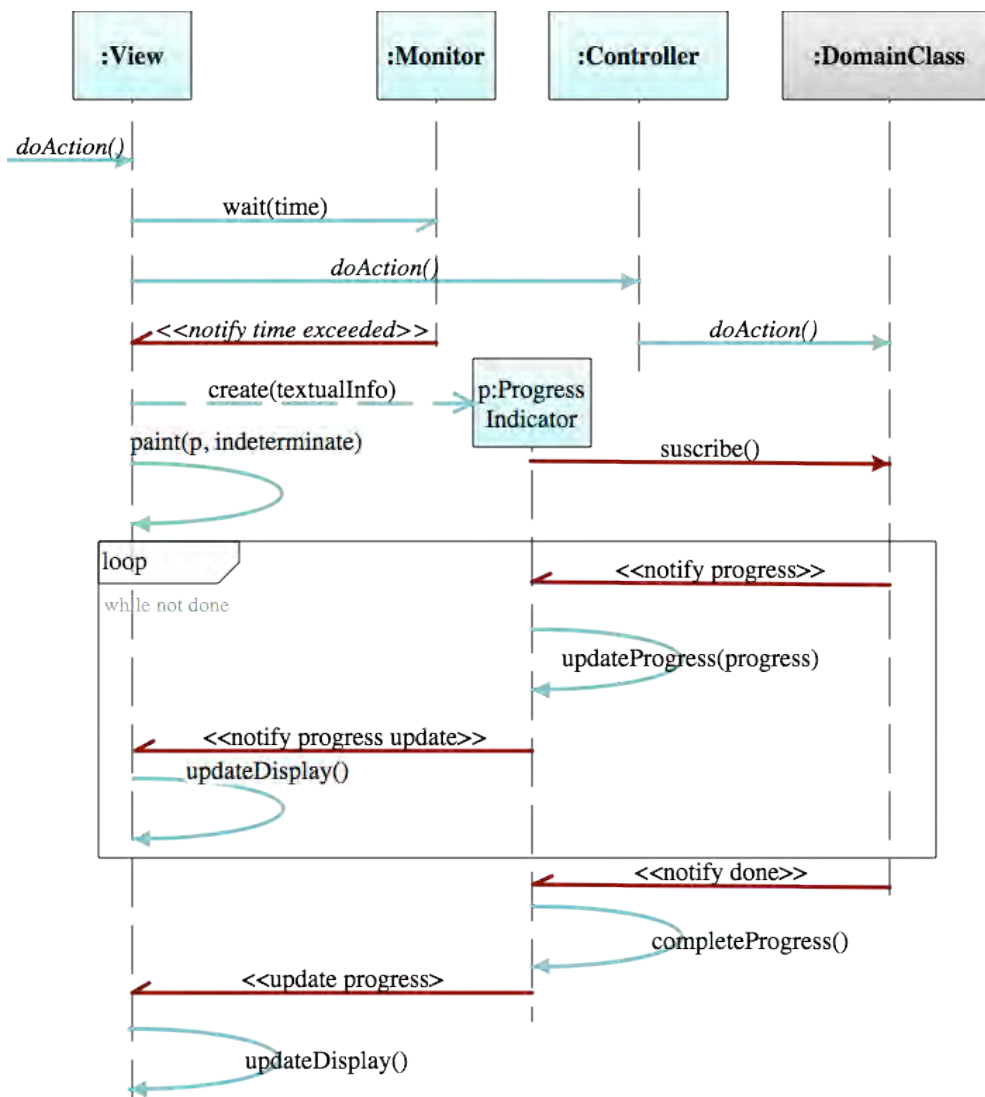


Figure 5.5-5: Sequence Diagram "Show Progress". Progress Feedback.

Classes and methods depicted in light blue represent that they belong to the Progress Feedback feature. Notifications in dark red represent Observer Pattern functionality. The gray DomainClass is a template class to be substituted at desing time by the appropriate system class containing the long action. For the full color legend see page 71.

## 5.6 “System Status Feedback” Usability Guideline for Software Development

The System Status Feedback Functional Usability Feature covers the need to provide the user with information on the different statuses the system might be in at any given time. An application can be in one or more statuses at once, so the user needs to be visually aware of all of them continuously.

The Usability Guideline for Software Development is made up of Analysis artifacts and Design artifacts. These are described for the the Abort feature in the following two sections.

### 5.6.1 Usability Guideline for Software Development: Analysis artifacts

There are four artifacts to be used during the analysis phase: the Usability Elicitation Guideline, the Usability Elicitation Clusters, the Usability Use Case Meta-models, the System Responsibilities for Usability. These artifacts are described in the following four sections.

#### 5.6.1.1 Usability Elicitation Guideline

Table 5.6-1 shows the Usability Elicitation Guideline for the System Status Feedback feature. In this guideline, there are three HCI recommendations, covering the notification features regarding system statuses, the different types of statuses and how they should be handled and displayed on screen. Below, all of these topics, as approached by this Usability Elicitation Guideline, are explained.

##### 5.6.1.1.1 Status change notification

HCI authors suggest that users must be notified when a change in system status occurs (SSF\_HCI-1). The system status can change because a user has requested the change, because an internal part of the software has done so, due to internal or external system failures, etc. All these sources of change must be considered upon elicitation (SSF\_ELAB-1). The stakeholders need to determine which statuses will be monitored and notified to the user upon change, as well s the possible sources of change (SSF\_Q-1 to SSF\_Q-2).

In Table 5.6-1, example SSF\_EX-1 “Browser ‘online’ status” describes an example for this HCI recommendation.

##### 5.6.1.1.2 Status types

HCI literature suggests different levels of obtrusivity depending on the criticality of a status change (SSF\_HCI-2). Critical information must be displayed obtrusively, requiring user acknowledgement, less critical information needs to be highlighted, drawing the user’s attention but not necessarily interrupting his work, and less important information can simply be displayed in the status area (SSF\_ELAB-2). Stakeholders must determine how critical each system status is, in order to display it appropriately (SSF\_Q-3).

Example SSF\_EX-2 “Status messages in Firefox” in Table 5.6-1 describes an example for this HCI recommendation.

##### 5.6.1.1.3 Status placement

HCI authors propose the center of the screen as the most effective place to position critical feedback, while top-left (in western culture) is also a prominent place to do so unobtrusively (SSF\_HCI-3). During elicitation, it must be determined if the system will need to accommodate more than one writing direction, as prominence of status areas will vary in each case (SSF\_ELAB-3). Finally, stakeholders must establish, for each type of system status, where its notifications will be displayed on screen (SSF\_Q-4).

Table 5.6-1. Usability Requirements Elicitation Guideline. System Status Feedback.

Identification			
Name	System Status Feedback (SSF)		
Family	Feedback		
Aliases	Status Display; Modeling Feedback Area		
Intent			
Providing the user with information on the different statuses the system might be in at any given time			
Problem			
An application can be in one or more statuses at once, so the user needs to be visually aware of all of them continuously.			
Context			
When changes that are important to the user occur or when failures that are important to the user occur: During task execution, because there are not enough system resources, because external resources are not working properly.			
Interrelationships			
Abort: Certain types of status feedbacks may need a 'cancel' option			
HCI Recommendation	Elaboration	Discussions with Stakeholders	Usage Examples (optional)
<p>SSF_HCI-1: Status changes</p> <p>HCI experts argue that the user wants to be notified when a change of status occurs.</p>	<p>SSF_ELAB-1 Source of status changing events</p> <p>Changes in the system status can be triggered by any of the following:</p> <ul style="list-style-type: none"> <li>- User-initiated events</li> <li>- Internal actions realizadas por el propio sistema</li> <li>- System failures</li> </ul> <p>Problems with external and internal resources</p>	<p>SSF_Q-1 Of which system statuses (and status changes) will the user be notified?</p> <p>SSF_Q-2 Which status changes are initiated by the user and which are initiated by the system or other resources ?</p>	<p>SSF_EX-1: Browser 'online' status</p> <p>Without proper status feedback a user may lose track of the actions he is allowed to perform at a given time. For example, the Online Status in a browser determines if the user is allowed to navigate only through cached pages (offline) or also through live web pages (online).</p>
<p>SSF_HCI-2: Status types</p> <p>Well-designed displays of the information to be shown should be chosen. They need to be unobtrusive if the information is not critically important, but obtrusive if something important happens.</p> <p>Displays should be put together in a way that emphasizes the important things, de-emphasizes the trivial, doesn't hide or obscure anything, and prevents one piece of information from being confused with another. They should never be re-arranged, unless users do so themselves. Attention should be called to important information with bright colours, blinking or motion, sound or all three – but a technique appropriate to the actual importance of the situation to the user should be used.</p>	<p>SSF_ELAB-2 Status types: Levels of importance</p> <p>For each piece of status information to be displayed, discuss with the user what type of information it is according to the following criteria:</p> <ul style="list-style-type: none"> <li>- Critical information needs to be displayed obtrusively</li> <li>- Important yet non-critical information needs to be highlighted</li> <li>- Less important information should be displayed in the status area</li> </ul> <p>During elicitation, the discussion of the exact response type can be left until interface design time, but the importance of the different situations and the general type of salience (obtrusive, highlighted or standard) that will be provided does need to be discussed at this stage.</p> <p>Overloading the user with too many obtrusive status notifications can be counterproductive, as can be undermining critical status changes by relegating them to the status area.</p>	<p>SSF_Q-3 For each system status, which type of notification should be shown to the user?</p>	<p>SSF_EX-2: Status messages in Firefox</p> <p><i>Critical:</i> When attempting to open a website without a live internet connection, Firefox will display an obtrusive message informing the user of this fact.</p> <p><i>Non-critical:</i> When the browser is set to a site that has been marked as a 'favorite' by the user, a small star will appear next to its URL.</p> <p><i>Less important:</i> When the browser has finished loading a page, the word "Done" will appear in the lower-left corner of the status bar.</p>

Identification			
<p>SSF_HCI-3: Status placement</p> <p>HCI literature mentions that users want one place where they know they can easily find the status information. Aside from the spot on the screen where users work, they are most likely to see feedback in the center or at the top of the screen, and least likely to notice it at the bottom.</p> <p>The standard practice of putting information about changes in state on a status line at the bottom of a window is particularly unfortunate, especially if the style guide calls for lightweight type on a grey background. The positioning of an item within the status display should be used to good effect. People born into a European or American culture tend to read left-to-right, top-to-bottom, and that something in the upper left corner will be looked at most often.</p>	<p>SSF_ELAB-3 Status placement and writing directions</p> <p>Ask about the users' reading direction and whether or not the system will need to accommodate more than one direction. Display areas that are prominent in one culture will be less so in others of different writing direction.</p>	<p>SSF_Q-4 How and where will each type of notification be shown to the user?</p>	<p>SSF_EX-3: Facebook in Hebrew</p> <p>When the Facebook application is set to the Hebrew language (and other languages with right-to-left script), every status area is mirrored horizontally. (i.e. the small red balloon that indicates 'new messages' will be displayed in the bottom-left corner of the page instead of to the bottom-right)</p>

### 5.6.1.2 Usability Elicitation Cluster Map

The Usability Elicitation Guideline in Table 5.6-1 suggests four discussions items (SSF\_Q-1 to SSF\_Q-4) to be held with stakeholders in order to elicit all aspects of the System Status Feedback feature. These discussion items can be clearly divided into three initial groups, or clusters, as described in the Usability Elicitation Cluster Map, shown in Figure 5.6-1 according to the portion of the System Status Feedback functionality that they cover.

**SSF\_EC-1 Knowing system statuses and their changes:** The discussion items in this cluster are aimed at determining the list of statuses that the system will need to keep track of and, for each of those statuses, the manners in which they can change. For example one status in a connectivity application could be “connection status”, which could change to and from ‘online’ and ‘offline’

**SSF\_EC-2 Handling status changes:** This cluster delves into the discussions regarding how the system is expected to react if/when the status changes mentioned above occur.

**SSF\_EC-3 Presenting system status notifications to user:** The discussion items included in this cluster deal with the placement and obtrusivity level of the notifications regarding each status change.

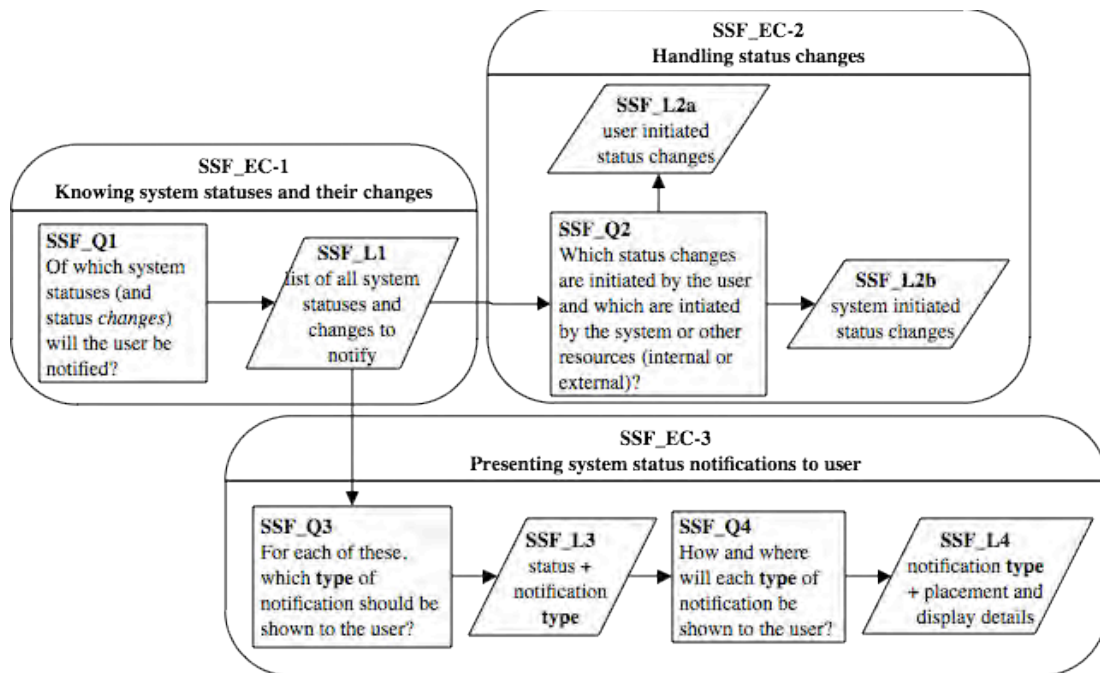


Figure 5.6-1: Elicitation Clusters. System Status Feedback

### 5.6.1.3 Use Case Meta-model

The Use Case Meta-model for the System Status Feedback Functional Usability Feature is shown in Figure 5.2-2, in which four use cases are identified and described below.

**SSF\_UC-1 ChangeStatus:** This use case begins when the system is requested to change the value (i.e. ‘online’ to ‘offline’) of a particular system status (i.e. ‘connection status’). Upon reception of this request, the system changes the status value and typically displays it (i.e. connection icon going from green to red) and performs an associated action (i.e. blocking internet access). Use cases DisplayStatus and SystemAction’ represent these last two actions respectively and are described below. Furthermore, status changes can be triggered by the user (User actor) or by the system itself (System actor).

**SSF\_UC-2 DisplayStatus:** Whenever a status is modified, the system must visually display the change for the user through the GUI accordingly.

**SSF\_UC-3 SystemAction\*:** As mentioned in the ChangeStatus use case description, this other system action is what should be performed by the system once a status has been changed, beyond the display of the change. For example, turning off the Wireless connection in a laptop not only changes the status icon from (📶) to (📴), it also performs a series of calls to the operating system to actually block hardware access to Wi-Fi networks.

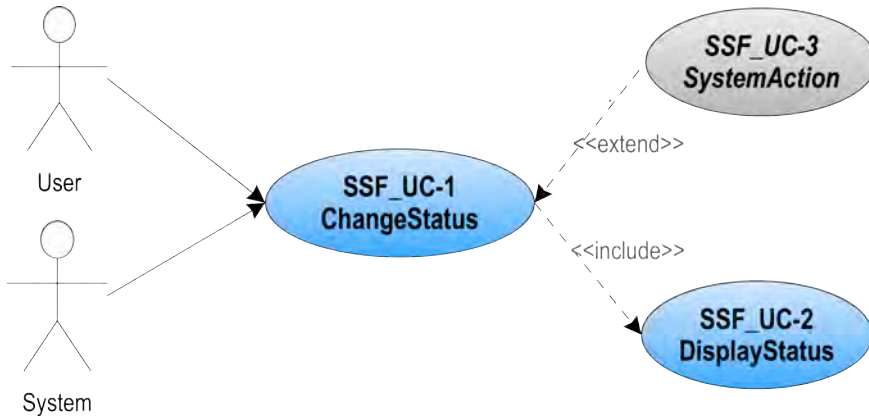


Figure 5.6-2 Use Case Meta-model. System Status Feedback

Use cases depend on one another as shown in Table 5.6-2, where we can see the following:

- The **ChangeStatus** use case needs the **SystemAction\*** use case (conditionally, hence the asterisk). This relationship only applies in the cases when changing a status will trigger a system action beyond updating the interface, as explained earlier.
- The **DisplayStatus** use case needs the **ChangeStatus** use case in order to be viable, because a change must occur in order for it to be displayed.

Table 5.6-2 Usability Use Case Dependencies: System Status Feedback Functional Usability Feature

	SSF_UC-1 ChangeStatus	SSF_UC-2 DisplayStatus	SSF_UC-4 SystemAction*
SSF_UC-1 ChangeStatus	-		X*
SSF_UC-2 DisplayStatus	X	-	
SSF_UC-3 SystemAction	X		
SSF_UC-4 SystemAction*			-

#### 5.6.1.4 System Responsibilities for Usability

Table 5.6-3 shows the proposed System Responsibilities for Usability for the present feature.

Table 5.6-3 System Responsibilities List for System Status Feedback

<b>System Responsibilities List for System Status Feedback</b>
SSF_SR-1 Be aware of system statuses (and their changes) The system must monitor all statuses for changes
SSF_SR-2 Handle user-initiated status changes The system must provide users with a way to change the statuses that require it
SSF_SR-3 Handle system-initiated status changes The system must provide means for predetermined external sources, or other parts of the system, to change statuses as required.
SSF_SR-4 Present system status notifications to users The system must notify users appropriately of each status change

These System Responsibilities for Usability are derived from the Usability Elicitation Clusters as follows:

**SSF\_EC-1 Knowing system statuses and their changes:** This cluster determines which statuses the system will need to be aware of, as well as the ways in which they can change. This elicitation cluster yields a single System Responsibilities, namely **SSF\_SR-1 Be aware of system statuses and their changes.**

**SSF\_EC-2 Handling status changes:** This cluster contemplates determining whether each status change is initiated by the user or by a third party (be it the system itself, other resources, etc) and what to do about them. This makes way for two system responsibilities **SSF\_SR-2 Handle User-initiated status changes** and **SSF\_SR-3 Handle system-initiated status changes**

**SSF\_EC-3 Presenting system status notifications to user:** Once a status change has occurred, regardless of source, the user must be notified appropriately. This cluster is formed by discussion items that deal with the type of notification to give for each status change as well as the on-screen characteristics of each.

Table 5.6-4 maps the relationships between the Usability Elicitation Clusters and the Usability System Responsibilities described above, for easy reference. Any project determined to require a specific Elicitation Cluster will also require its related System Responsibilities. Likewise, if a cluster is discarded during elicitation, its related responsibilities will not be a part of the resulting system. In the case of Abort this relationship is one-to-one.

Table 5.6-4 Usability Elicitation Clusters / System Responsibilities for Usability Mapping for System Status Feedback

Elicitation Clusters	Dependent Responsibilities
SSF_EC-1 Knowing system statuses and their changes	SSF_SR-1 Be aware of system statuses (and their changes)
SSF_EC-2 DisplayStatus	SSF_SR-2 Handle user-initiated status changes SSF_SR-3 Handle system-initiated status changes
SSF_EC-3 SystemAction	SSF_SR-4 Present system status notifications to users

As was the case with use case interrelationships, the ChangeStatus use case (SSF\_UC-1) has the highest possible number of System Responsibility dependencies, due to the fact that it is the core of the feature.

## 5.6.2 Usability Guideline for Software Development: Design artifacts

In order to support the System Responsibilities at design level, the following sections describe the suggested High-level Design Components for the Progress feature.

### 5.6.2.1 High-level Design Component Responsibilities

In order to support the System Responsibilities at design level, the following sections describe the suggested High-level Design Components for the System Status Feedback feature, summarized in Table 5.6-5.

#### 5.6.2.1.1 UI Component

This component is responsible for capturing all user invocations and forwarding them (possibly through a delegating component) to the appropriate part of the domain, usually that responsible for executing the invoked action. Specifically for this feature, calls from the user to change a system status are captured by the UI Component. The UI Component is also responsible for displaying all status information appropriately, as well as any changes of which it may be notified.

#### 5.6.2.1.2 Status Manager Component

This component is responsible for monitoring the system for status-altering actions. When one is detected, it is forwarded to the appropriate Status Component.



### 5.6.2.1.3 Status Component

This component is the one responsible for holding all the information related to a single status. It holds all possible values ('online', 'idle', 'busy', 'offline') for a single status ('connection status') as well as the active value ('online').

### 5.6.2.1.4 Domain Component

A Domain Component represents the part of the system that is ultimately responsible for executing the actions requested by the user.

Table 5.6-5: Usability Guideline: High-level Design Component Responsibilities. System Status Feedback

System Responsibility	High-level Design Component Responsibilities
SSF_SR-1 Be aware of system statuses (and their changes)	<p>Certain <i>Domain Components</i> can execute actions that will change one or more application statuses.</p> <p>A <i>StatusManager Component</i> is responsible for monitoring said <i>Domain Components</i> and listen for their status-altering actions.</p> <p>A <i>Status Component</i> is responsible for holding all the information relating to a particular status and for modifying it according to <i>StatusManager</i> orders. All <i>Status Components</i> can have one active status value at any given time (i.e. "online status" can be 'online', 'idle', 'busy', 'offline', etc.).</p> <p>The component responsible for handling user events (<i>UI</i>) must monitor all <i>Status Components</i> and notify the user of any changes.</p>
	<p>The component responsible for handling user events (<i>UI</i>) listen for user actions and order their execution</p> <p>The component in charge of delegating actions (if any) is responsible for ordering the appropriate <i>Domain Component</i> to execute said action.</p> <p>Upon execution of actions that are status-changing, each <i>Domain Component</i> is responsible for notifying any interested parties (specifically the <i>Status Manager Component</i>, in this case)</p> <p>The <i>StatusManager</i> component then forwards the updated information onto the appropriate <i>Status Component</i>.</p> <p>Said <i>Status Component</i> is then responsible for determining the effect, if any, that the received information will have on its current active status value. It will, when applicable, change said value and notify any interested parties (specifically the <i>UI Component</i> in this case)</p> <p>The <i>UI Component</i> will update the status display for every notification of status change received.</p>
SSF_SR-2 Handle user-initiated status changes	<p>Upon execution of actions that are status-changing--invoked by any other class in the system or an external source--each <i>Domain Component</i> is responsible for notifying any interested parties (specifically the <i>Status Manager Component</i>, in this case), as is the case when such an action is invoked by the user through the UI.</p> <p>The <i>StatusManager</i> component then forwards the updated information onto the appropriate <i>Status Component</i>.</p> <p>Said <i>Status Component</i> is then responsible for determining the effect, if any, that the received information will have on its current active status value. It will, when applicable, change said value and notify any interested parties (specifically the <i>UI Component</i> in this case)</p> <p>The <i>UI Component</i> will update the status display for every notification of status change received.</p>
SSF_SR-3 Handle system-initiated status changes	<p>The <i>UI Component</i> is responsible for knowing how and where each status (and its possible values) are displayed within the interface, and thus update it accordingly upon reception of notifications of status value change.</p>

### 5.6.2.2 Low-level Design Component Responsibilities for MVC

When instantiating for a MVC architecture, the UI Component is instantiated by View the object(s) and takes over all of its responsibilities except for the delegation of actions to other objects. This responsibility falls on the **Controller** object(s) of the MVC architecture.

The Status Component is represented by the **Status** object. However, each individual system status is represented by one concrete object that which will inherit common attributes and functionality from **Status**. The **Status Managing** component is represented by a **StatusManager** object, taking over all of its responsibilities.

Table 5.6-6 details the Low-level Design Component Responsibilities described above and how they carry out each of the System Responsibilities defined earlier. For each System Responsibility, the sequence of actions required by the different objects is presented as well as a set of UML diagrams that depict each of these interactions.

Table 5.6-6: Usability Guideline: Low-level Design Component Responsibilities (MVC). System Status Feedback.

System Responsibility	Objects					Fig
	View	Controller	StatusManager	Status	DomainClass	
SSF_SR-1 Be aware of system statuses (and their changes)			1. Upon system initialization, the <i>StatusManager</i> subscribes to each <i>DomainClass</i> which it knows can execute status-changing actions.		2. The <i>DomainClass</i> represents the domain object(s) responsible for executing actions that lead to system state changes. It must notify all subscribers ( <i>StatusManager</i> ) of any changes.	Figure 5.6-4
	1. The <i>View</i> must subscribe to each <i>Status</i> object upon system initialization.			2. The <i>Status</i> object holds all the information related to one system status and the means to change and query this information. It must notify all subscribers ( <i>View</i> ) of any changes.		
SSF_SR-2 Handle user-initiated status changes	1. The <i>View</i> listens user's requests for execution actions action, and forwards it to the <i>Controller</i> .	2. The <i>Controller</i> orders the appropriate <i>DomainClass</i> to execute said actions	4. The <i>StatusManager</i> determines the corresponding <i>Status</i> object to update and does so with the information sent forth by the <i>DomainClasses</i>	5. The <i>Status</i> calculates the effect, if any, that the received information has on its current active status value, change it, if applicable, and notify its subscribers ( <i>View</i> )	3. The <i>DomainClass</i> executes the (status-altering) action and for notifies the <i>StatusManager</i>	
SSF_SR-3 Handle system-initiated status changes			2. The <i>StatusManager</i> determines the corresponding <i>Status</i> object to update and does so with the information sent forth by the <i>DomainClasses</i>	3. The <i>Status</i> calculates the effect, if any, that the received information has on its current active status value, change it, if applicable, and notify its subscribers ( <i>View</i> )	1. The <i>DomainClass</i> executes the (status-altering) action--triggered by a foraneous resource or other parts of the system--and for notifies the <i>StatusManager</i>	
SSF_SR-4 Present system status notifications to users	The <i>View</i> knows which type of status notification to give for each status change. It also knows how and where to display each type of status notification and does so upon notification of <i>Status</i> objects.					

### 5.6.2.3 Usability Software Design Meta-models

This section describes the UML diagrams representing the Low-level Design Component Responsibilities described above. Below, the class diagram is described, as well as the classes involved in this feature and their interrelationships, followed by the descriptions of the sequence diagram.

#### 5.6.2.3.1 Class Diagram

Figure 5.6-3 below shows the class diagram for the System Status Feedback Functional Usability Feature. The main objects involved are the View, Controller, Status, ConcreteStatus and StatusManager. The first two, fulfilling their role within MVC, respectively capture and distribute the user calls to perform actions.

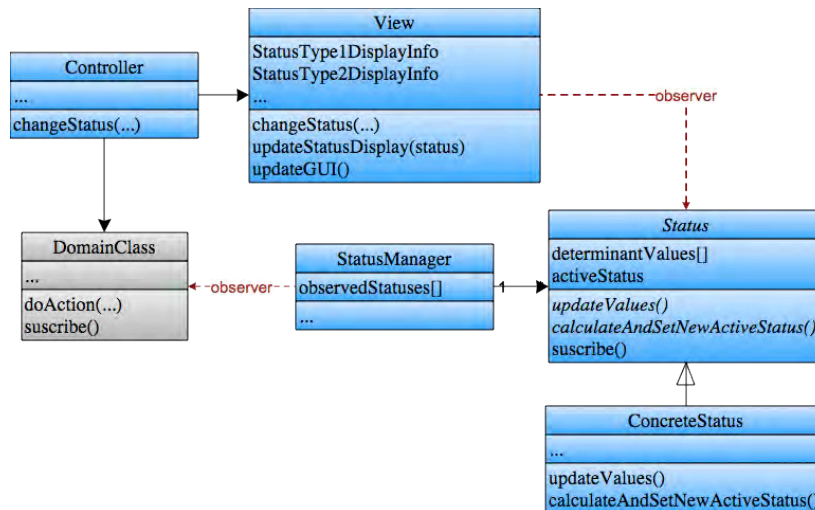


Figure 5.6-3: Usability Design Meta-model. Class diagram. System Status Feedback.

For each status that is defined for the system, one concrete class of the abstract Status class must be created. For example, in a browsing system, the design would have one abstract Status class and also a child ConnectionStatus class. It is this child (concrete) class which will hold all the information regarding that particular status (the connection status) and perform any changes to its active value (online, offline, etc) upon request.

Finally, the StatusManager controls all the Statuses and subscribes to the DomainClasses that may request changes in them at any time.

#### 5.6.2.3.2 Sequence Diagram "Change Status"

Figure 5.6-4 shows the one sequence diagram for changing a single system status. It covers all Low-level Design Component Responsibilities for this feature.

The sequence starts in one of two alternative ways: In a first scenario, it may start with a user initiated request to change a particular status. This call is forwarded to the Controller who is responsible for determining any other actions that might need to take place before changing the status value. To do so, it locates the proper DomainClass and orders it to execute the requested action. In a second scenario, the call for DomainClass to execute said action comes from an outside source, beyond the scope of this sequence diagram.

In either case, once the action has been executed, the DomainClass informs its observers (of which StatusManager is one) of the change. With this information, the StatusManager locates the appropriate Status class and orders it to change accordingly. Once it does so, the Status class notifies the View of the change and the View updates its displays and any other part of the GUI as it may deem necessary.

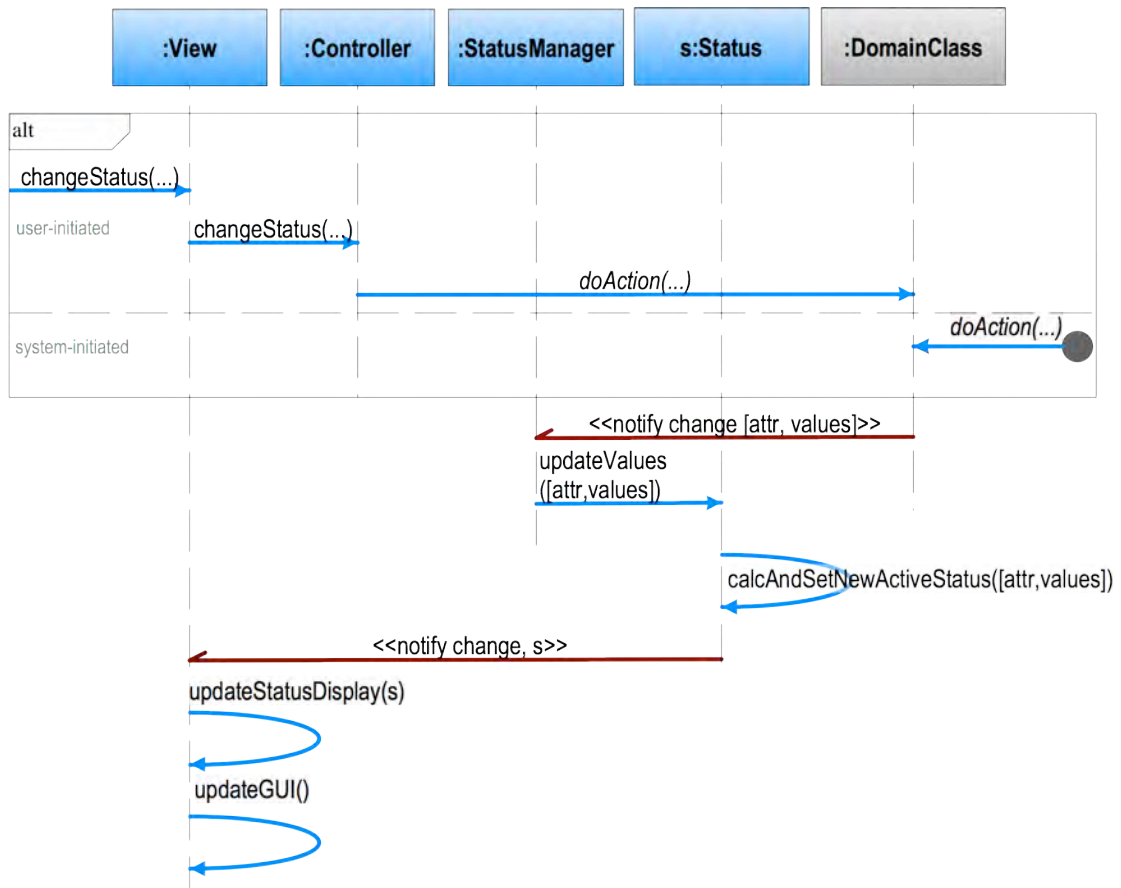


Figure 5.6-4: Sequence Diagram "Change Status". System Status Feedback.

Classes and methods depicted in blue represent they belong to the System Status Feedback feature. Notifications in dark red represent Observer Pattern functionality. The gray DomainClass is a template class to be substituted at desing time by the appropriate system class. For the full color legend see page 71.

## 5.7 “Warning” Usability Guideline for Software Development

The Warning Functional Usability Feature covers the user’s need to provide different alert types upon execution of sensitive actions. The main goal of the Warning feature is to provide appropriate alerts upon execution of ‘potentially damaging’ actions.

The Usability Guideline for Software Development is made up of Analysis and Design artifacts, described for the Warning feature in the following two sections.

### 5.7.1 Usability Guideline for Software Development: Analysis artifacts

There are four artifacts to be used during the analysis phase: the Usability Elicitation Guideline, the Usability Elicitation Clusters, the Usability Use Case Meta-models, the System Responsibilities for Usability. These artifacts are described in the following four sections.

#### 5.7.1.1 Usability Elicitation Guideline

Table 5.7-1 shows the Usability Elicitation Guideline for the Undo Functional Usability Feature. In this guideline, there is a single HCI recommendation described below. It addresses the types of warnings that can be shown to the user, and in what cases each of these would be most appropriate.

##### 5.7.1.1.1 Types of Warnings

HCI authors suggest studying each possible action that may require a warning for characteristics like their potential degree of damage, reversibility, etc. Once this is established, authors suggest choosing between three types of warnings for each action: notifications (letting the user know that an action has taken place), confirmation (asking a user to ‘ok’ an action before execution), and authorization requesting user permission (usually through credentials) for execution of an action (W\_HCI-1). All three types of warnings should be use only when needed to avoid overwhelming the user with interruptions. (W\_ELAB-1) Stakeholder discussions should determine which actions are expected to trigger warnings and, for each, which kind of warning is to be used. Discussions should also revolve around how each type of warning is to be displayed for the user (W\_Q-1 to W\_Q-4).

In Table 5.7-1 W\_EX-1 through W\_EX-3 describe an example for this HCI recommendation.

Table 5.7-1. Usability Requirements Elicitation Guideline. Warning.

Identification			
Name	Warning		
Family	Feedback		
Aliases	Status Display; Modeling Feedback Area		
Intent			
Providing different alert types upon execution of 'potentially damaging' actions			
Problem			
Certain application tasks have potential serious consequences (that, for example, may not be undoable) so the application might need to verify with the user one last time before actually executing the task, to prevent them from calling said tasks by mistake, or to allow them to reconsider if needed.			
Context			
Applications where user tasks may have 'potentially damaging' effects, including permanent changes or loss of data			
Interrelationships			
Abort: Certain types of warnings require a 'cancel' button. See 'cancel operation' section in Abort Feature			
HCI Recommendation	Elaboration	Discussions with Stakeholders	Usage Examples (optional)
<p>W_HCI-1 Warning types</p> <p>For each action that a user may take, consider the following aspects:</p> <ul style="list-style-type: none"> <li>- the reversibility of the action</li> <li>-the proportion of reversible actions that the system supports</li> <li>- the frequency with which the action is taken</li> <li>-the degree of damage that may be caused</li> <li>-the immediacy of feedback</li> </ul> <p>to determine which of the following types of warning needs to be given to the user:</p> <ul style="list-style-type: none"> <li>- Notification</li> <li>- Confirmation</li> <li>- Authorization</li> </ul>	<p>W_ELAB-1 Warning considerations</p> <p>The Warnings addressed herein pertain to Notifications, Confirmations or Authorizations presented to the user during or before execution of an action</p> <p>Stakeholder should know that the more damaging the action (for irreversible actions) the higher the level of warning. (and vice-versa). Be careful not to over-do.</p> <p>Actions of little damage can be left "open" as to not overload the user with notifications.</p> <p>Use notifications only when actually useful (the user will do something with the information provided)</p> <p>Warnings are considered preemptive, so notification that an error has occurred falls outside of the scope of this pattern (See Status Feedback).</p>	<p>W_Q-1 Which user actions require the user to be warned?</p> <p>W_Q-2 Of these, which can't start execution until some sort of user acknowledgement takes place?</p> <p>W_Q-3 Of those that need approval, which are highly damaging/sensitive, therefore needing credential approval?</p> <p>W_Q-4 For every action mentioned, which information will be shown to the user?</p>	<p>W_EX-1 Notification</p> <p>"Remember that..." during or before execution of an action. It does not interrupt nor does it expect user feedback</p> <p>W_EX-2 Confirmation</p> <p>"Are you sure you want to...?" right before execution of damaging action. User needs to OK for execution to proceed</p> <p>W_EX-3 Authorization</p> <p>"You need to provide login and password before you can delete this file" right before execution of highly damaging action. User needs to provide credentials or otherwise be authorized before execution can proceed</p>

### 5.7.1.2 Usability Elicitation Cluster Map

The Usability Elicitation Guideline In Table 5.7-1 suggests four discussions items to be held with stakeholders in order to elicit all aspects of the Warning Functional Usability Feature. These discussion items can be clearly divided into three initial groups, or clusters, as described in the Usability Elicitation Cluster Map in shown in Figure 5.7-1, according to the portion of the Warning functionality that they cover.

**W\_EC-1 Awareness of sensitive actions:** This cluster contains one stakeholder discussion, aimed at determining which system actions will require warnings to be displayed.

**W\_EC-2 Determining appropriate warning per action:** Once the actions that will require warnings have been established, they must be studied individually to determine which type each will require. Stakeholders must first determine which of the actions will require the user to acknowledge it before executing it and which will not. Those that do not, will require a type of ‘notification’ warning, while those that do might require ‘authorization’ (an ‘ok’) or ‘authentication’ (user credential check before execution). At the end of this group of discussions, stakeholders must have a clear vision of how the sensitive actions are divided into these three groups.

**W\_EC-3 Displaying different types of warnings:** For each type of warning mentioned above, stakeholders must determine how they will be displayed on screen and what information will be shown to the user.

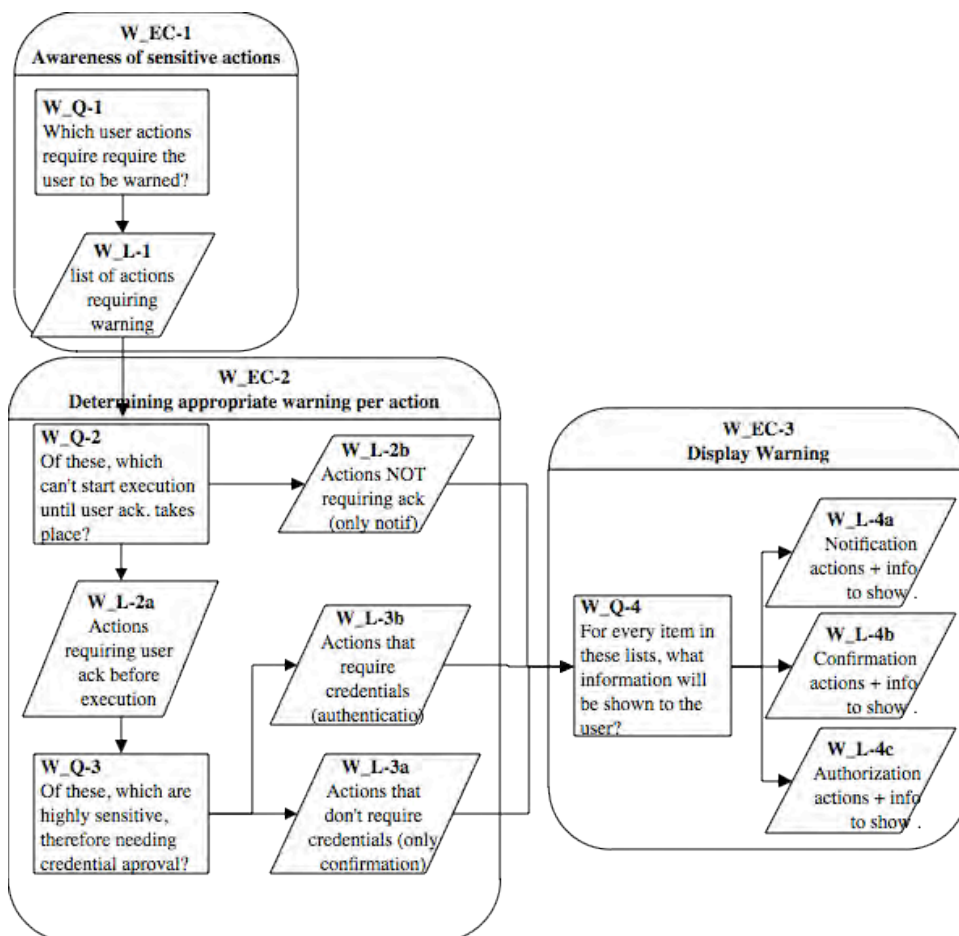


Figure 5.7-1: Elicitation Clusters. Warning.

### 5.7.1.3 Use Case Meta-model

The Use Case Meta-model for the Warning feature is shown in Figure 5.7-2 (See page 71 for color legend), in which seven use cases are identified and described below.

**W\_UC\_1 UserAction:** This template use case represents any user action in the system that may trigger a warning.

**W\_UC\_2 DisplayWarning:** Is the parent use case for displaying a warning. These use cases extend the UserAction, and may or may not be triggered within it as determined during elicitation.

For example, if an 'export video' feature in a video application must give a warning when exporting videos larger than 100MB in size, an 'export video' use case would trigger the appropriate warning only if this condition is met, and not otherwise.

**W\_UC\_3 DisplayAuthorization:** An authorization is the type of warning that requires the user not only to 'ok' an action, but also to introduce some form of identity verification data (as would be a login and password combination) to proceed.

**W\_UC\_4 DisplayConfirmation:** A confirmation is the type of warning where the user must only 'ok' an action before execution (i.e. 'Are you sure you want to empty the trash? Yes, No').

**W\_UC\_5 DisplayNotification:** A notification is simply a message relayed to the user about an action that has already taken place. The user has no means to stop the action, and is only being informed of it having been executed.

**W\_UC\_6 IdentifyUser:** This template use case represents the domain-specific use case in which the user introduces his credentials to be identified by the system. As this is inherent to each particular system, the actual use case must replace this template use case when designing the actual use case model. This use case is included within the Authorization warning.

**W\_UC\_7 Confirm:** Confirm is a use case included within the corresponding warning, that represents a user confirmation, which can be as simple as clicking an 'ok' button, but leaves the meta model open to more elaborate alternatives.

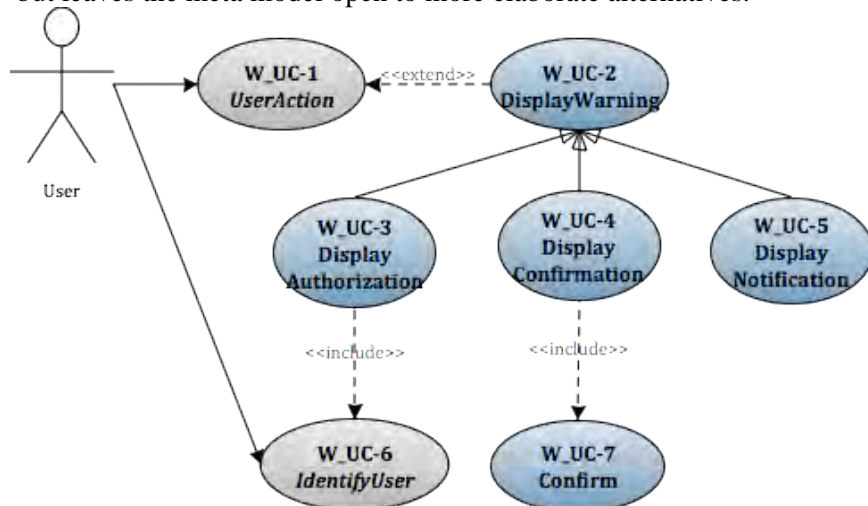


Figure 5.7-2 Use Case Meta-model. Warning

The applicability of each of these use cases will depend on the results of the elicitation process. If during elicitation of the Warning feature it is determined that, for example, user authorization (through the use of a login and password combination, for example) will never be needed, the DisplayAuthorization use case, as well as IdentifyUser, would be discarded. Use cases also depend on one another. These dependencies are shown in Table 5.7-2, where we can see the following:



- The **UserAction** use case, being a template use case representing just about any system action, needs no other use case to be viable.
- The **DisplayWarning** use case needs the **UserAction**, as a warning will only appear in the system when triggered by it.
- The **DisplayAuthorization** use case needs its parent use case, **DisplayWarning**, as well as the **UserAction**. It also needs the **IdentifyUser** use case to be viable, as it is in this use case that the actual user identification occurs (via login and password, for example)
- The **DisplayNotification** use case needs no more than its parent use case, **DisplayWarning** and the **UserAction** to be viable.
- The **IdentifyUser** use case, being a template use case representing the system's own way to identify users (for example, a login use case) needs no other use case within the scope of this feature.
- Finally, the Confirm use case needs the **DisplayConfirmation** use case to be viable, as a user will only be asked to Confirm execution of an action from within **DisplayConfirmation**.

Table 5.7-2 Usability Use Case Dependencies: Warning Functional Usability Feature

	W_UC-1 User Action	W_UC-2 Display Warning	W_UC-3 Display Authorization	W_UC-4 Display Confirmation	W_UC-5 Display Notification	W_UC-6 Identify User	W_UC-7 Confirm
W_UC-1 User Action	-						
W_UC-2 Display Warning	X	-					
W_UC-3 Display Authorization	X	X	-			X	
W_UC-4 Display Confirmation	X	X		-			X
W_UC-5 Display Notification	X	X			-		
W_UC-6 Identify User						-	
W_UC-7 Confirm				X			-

By looking at the columns of Table 5.2-2, up to any two of the three types of Warnings could be discarded and the feature would still be viable (i.e. a system that only requires notifications, discarding the use cases related to authorization and confirmation).

#### 5.7.1.4 System Responsibilities

Table 5.7-3 shows the proposed System Responsibilities for the Warning feature.

Table 5.7-3 System Responsibilities List for Warning

System Responsibilities List for Undo
W_SR-1 Be aware of damaging actions The system must know which actions will require warnings
W_SR-2 Notify The system must be aware of which of those actions require notifications
W_SR-3 Request confirmation The system must be aware of which of those actions require the user to confirm before execution
W_SR-4 Request authorization The system must be aware of which of those actions require the user to be properly authorized before execution
W_SR-5 Display warning The system must know which warning information to show for each action.

These System Responsibilities for Usability are derived from the Usability Elicitation Clusters as follows:

**W\_EC-1 Awareness of sensitive actions:** As mentioned earlier, this elicitation cluster contemplates determining which actions will require some kind of warning to be displayed. It yields a single System Responsibility, **W\_SR-1 Be aware of damaging actions**.

**W\_EC-2 Determining appropriate warning types per action:** This elicitation cluster contains the discussion items related to separating the actions determined to need warnings into three groups: those that need notification, confirmation and

authentication. Naturally, this leads to three System Responsibilities, one for each type of warning, as they are all handled differently by the system. These are **W\_SR-2 Notify**, **W\_SR-3 Request Confirmation** and **W\_SR-4 Request Authorization**.

**W\_EC-3 Displaying Warning:** This cluster contains a single discussion item regarding how warnings will be displayed and what information they will contain, yielding the System Responsibility **W\_SR-5 DisplayWarning**.

Table 5.7-4 maps the relationships between the Usability Elicitation Clusters and the Usability System Responsibilities described above, for easy reference. Any project determined to require a specific Elicitation Cluster will also require its related System Responsibilities. Likewise, if a cluster is discarded during elicitation, its related responsibilities will not be a part of the resulting system.

Table 5.7-4 Elicitation Clusters / System Responsibilities Mapping for Warning

Use Cases	Dependent Responsibilities
W_EC-1 Awareness of sensitive actions	W_SR-1 Be aware of damaging actions
W_EC-2 Determining appropriate warning types per action	W_SR-2 Notify W_SR-3 Request confirmation W_SR-4 Request authorization
W_EC-3 Displaying warning	W_SR-5 DisplayWarning

## 5.7.2 Usability Guideline for Software Development: Design artifacts

The design artifacts of the Usability Guideline for Software Development for the Warning feature are described in the following sections. The System Responsibilities are brought to a lower abstraction level as High-level Design Component Responsibilities in section 5.7.2.1. Section 5.7.2.2 expresses them as Low-level Design Component Responsibilities (for a MVC architecture). Section 5.7.2.3 presents the Usability Design Meta-models for said Low-level Design Component Responsibilities as object-oriented class and sequence diagrams.

### 5.7.2.1 High-level Design Component Responsibilities

In order to support the System Responsibilities at design level, the following sections describe the suggested High-level Design Components for the Warning feature.

#### 5.7.2.1.1 User Interface (UI) Component

This component is responsible for capturing calls for action execution that might trigger user warnings. It is also the responsibility of the UI to know how to display warnings, for example: notification warnings could be dialogue boxes with text and an OK button, authentication warnings could be dialogue boxes with text and two buttons: OK and Cancel, and authorization warnings could follow the same logic with the addition of asking for user authorization (login and password, for example). The UI Component must also relay information to the user, including feedback after an action has been executed. Within this usability feature, such information entails displaying the warnings themselves, and requesting user acknowledgement or further information (i.e. login and password) as needed.

#### 5.7.2.1.2 Warning Component

The Warning component represents a placeholder for the information that is to be shown the user requesting a sensitive action. As briefly introduced for the previous component, the this placeholder be of three different types: notification warnings, for actions for which the user must only be notified, authorization warnings, for those needing to be okayed by the user, and authentication warnings, for actions needing credential recognition before execution.

These three types of warnings have different structures and hold different types of information, according to the level of sensitivity of the action that triggers them. These structures are recognized by the View, as mentioned earlier, and displayed uniquely.

### 5.7.2.1.3 Domain Component

As in most other usability features, the Domain Component represents the part of the system that is ultimately responsible for executing the actions requested by the user.

### 5.7.2.1.4 Wrapping Component

This component is a Wrapper as defined by [37] for the Domain Component. It is responsible for knowing which type of warning is required for each sensitive action of its wrapped Component and what information must go into it. DomainClassWrap is also responsible for creating the Warning object of the appropriate type with the appropriate information when a call to execute a sensitive action is received. For example, for a mailing system containing a Mailer class, if Mailer contains sensitive methods, Mailer should be renamed to MailerPostfix (i.e. MailerCore, MailerDomain) and a new class called Mailer should be created. This new class is a wrapper to MailerPostfix and is in charge of determining whether or not a warning will be needed (and if so, for triggering it) for the invoked action).

Table 5.7-5: Usability Guideline: High-level Design Component Responsibilities. Warning

System Responsibility	Generic Component Responsibilities
W_SR-1 Be aware of damaging actions	<p>For each Domain Component that includes at least one 'damaging' method (one that needs to incur in a warning of some kind), a Wrapping Component must exist. This Wrapping Component mimics the structure of the Domain Component (it must have the same number of methods and the same method names as Domain Component), and it will 'sit' between any invoking class and said DomainComponent. All methods in a Wrapping Component consist of a) a flag check, to determine if it is safe to invoke the method of the same name in the DomainComponent, and, b) a call to said method in Domain Component.</p> <p>For example, if a component called 'SalesItem' is determined to have a 'damaging' method, the component will be renamed to, for example, 'SalesItemDomain', to allow for the creation of its Wrapping Component, which must be called 'SalesItem' for transparency's sake (an invoking class need not know if it's dealing with a Wrapper or a Domain Component).</p> <p>When invoked, methods in the Wrapping Component can respond (to their invokers) that it is not yet safe to invoke the requested method, and an appropriate Warning is issued. It is the Wrapping Component who is responsible for knowing which method call triggers which kind of warning (Notification, Confirmation, Authorization) and for determining whether or not the invocation of the method is safe.</p>
W_SR-2 Notify	Once the Warning is issued, if it is of the kind Notification, it must reach the UI Component, after which the invocation automatically returns to the Wrapping Component for execution. Since it is now safe to invoke the action in the DomainClass, the Wrapping Component does so.
W_SR-3 Request confirmation	If the issued Warning is a Confirmation, it must also reach the UI Component but in this case it must wait for the user to 'OK'. Once s/he does so, the invocation returns to the Wrapping Component for execution. Since it is now safe to invoke the action in the DomainClass, the Wrapping Component does so.
W_SR-4 Request authorization	Finally, if the issued Warning is an Authentication, it must reach the UI Component, which will then go through all the necessary steps (outside of the scope of this pattern) to perform the necessary credential cross-checking. Once the user has been authenticated in this manner, the call will return to Wrapping Component for execution. Since it is now safe to call the action in the DomainClass, the Wrapping Component does so.
W_SR-5 Display warning	The UI Component is responsible for displaying the Warning information and for receiving and processing the necessary user input to satisfy the given Warning (if applicable)

### 5.7.2.2 Low-level Design Component Responsibilities for MVC

When instantiating for a MVC architecture, the UI Component is instantiated by View the object(s) and takes over all of its responsibilities except for the delegation of actions to other objects. This responsibility falls on the **Controller** object(s) of the MVC architecture.

The Warning Component is represented by the **Warning** object, and its three children **Notification**, **AuthorizationRequest** and **AuthenticationRequest**, depending on the type of warning that is required.

The Wrapping Component and Domain Component are represented by the **DomainClassWrap** and the **DomainClass** respectively.

Table 5.7-6 details the Low-level Design Component Responsibilities described above and how they carry out each of the System Responsibilities. For each System Responsibility, the sequence of actions required by the different objects is presented as well as a set of UML diagrams that depict each of these interactions.

Table 5.7-6: Usability Guideline: Low-level design component responsibilities (MVC). Warning.

System Responsibility	Objects					Fig
	View	Controller	DomainClassWrap	DomainClass	Warning	
W_SR-1 Be aware of damaging actions	1. View listens for user calls to actions, <code>doAction()</code> , and passes them on to the Controller	2. The Controller forwards the call to <code>doAction()</code> to the appropriate class. Controller is not aware of the existence of <i>DomainClassWraps</i> , it simply forwards the call to the class it know to be responsible for handling the method ( <i>DomainClassWraps</i> take on the original name of the <i>DomainClass</i> )	3. If a <i>DomainClassWrap</i> exists for the <i>DomainClass</i> the Controller is trying to reach, it will be the one receive the method call, which will invoke its own implementation of <code>doAction()</code> , 4. In it, it will then check if the called method is 'OK' to execute with <code>checkOK('doAction')</code> . 5. If it is, it will call onto <i>DomainClass</i> to execute the method 6b. Otherwise it will create a Warning of the appropriate type (Notification, Authentication or Confirmation) with the information pertaining to the invoked method	6a. <i>DomainClass</i> executes the invoked method, <code>doAction()</code> .		Figure 5.7-4
W_SR-2 Notify	2. When the View receives a <i>Notification</i> , it displays a message to the user with the info contained within the <i>Notification</i> object. No user feedback is expected, as the control is returned to the <i>DomainClassWrap</i> via the Controller	3. The Controller requests <i>DomainClassWrap</i> to set the flag for 'doAction' to 'OK'. 4. It then calls the <code>doAction()</code> method on <i>DomainClassWrap</i> again, prompting a re-check of the flag for the current method.	5. Since the flag is now set to 'OK', the <i>DomainClassWrap</i> immediately forwards the method call to <i>DomainClass</i>	6. <i>DomainClass</i> executes the invoked method, <code>doAction()</code> .	1. The new Warning issued (in this case a <i>Notification</i> ) is forwarded onto the View	
W_SR-3 Request confirmation	2. When the View receives a <i>Confirmation</i> , it displays a dialogue to the user with the info contained within the <i>Notification</i> object and the option to 'OK' or 'Cancel'. If the user chooses to 'OK', the control is returned to the <i>DomainClassWrap</i> via the Controller	3. The Controller requests <i>DomainClassWrap</i> to set the flag for 'doAction' to 'OK'. 4. It then calls the <code>doAction()</code> method on <i>DomainClassWrap</i> again, prompting a re-check of the flag for the current method.	5. Since the flag is now set to 'OK', the <i>DomainClassWrap</i> immediately forwards the method call to <i>DomainClass</i>	6. <i>DomainClass</i> executes the invoked method, <code>doAction()</code> .	1. The new Warning issued (in this case a <i>Confirmation</i> ) is forwarded onto the View	
W_SR-4 Request authorization	2. When the View receives an <i>Authentication</i> , it engages in the appropriate actions to identify the current user (perhaps involving other relevant domain classes) Once the user has been properly identified by the system, the control is returned to the <i>DomainClassWrap</i> via the Controller	3. The Controller requests <i>DomainClassWrap</i> to set the flag for 'doAction' to 'OK'. 4. It then calls the <code>doAction()</code> method on <i>DomainClassWrap</i> again, prompting a re-check of the flag for the current method.	5. Since the flag is now set to 'OK', the <i>DomainClassWrap</i> immediately forwards the method call to <i>DomainClass</i>	6. <i>DomainClass</i> executes the invoked method, <code>doAction()</code> .	1. The new Warning (in this case an <i>Authentication</i> ) is forwarded onto the View	
W_SR-5 Display warning	It is the View's responsibility to display Warnings. To do so appropriately, it uses all the information available in the <i>Warning</i> object.					

### 5.7.2.3 Usability Software Design Meta-models

These UML diagrams represent the Low-level Design Component Responsibilities described in earlier. The following sections describe the class diagram and the classes involved in this feature and their interrelationships, followed by the descriptions of the sequence diagrams.

#### 5.7.2.3.1 Class Diagram

Figure 5.7-3 below shows the class diagram for the Warning Functional Usability Feature. As described in the Low-level Design Component Responsibilities Table (see Table 5.2-6), the main objects involved are the View, Controller, Warning, Notification, ConfirmationRequest, AuthorizationRequest, DomainClass and DomainClassWrap. The first two, fulfilling their role within MVC, respectively capture and distribute the user calls to perform actions. The Warning class (and its children) controls the contents of a warning, the Domain class holds the methods that are to be executed and the DomainClassWrap sits between the DomainClass and the rest of the system, filtering calls to sensitive methods and issuing the appropriate warnings for them when needed.

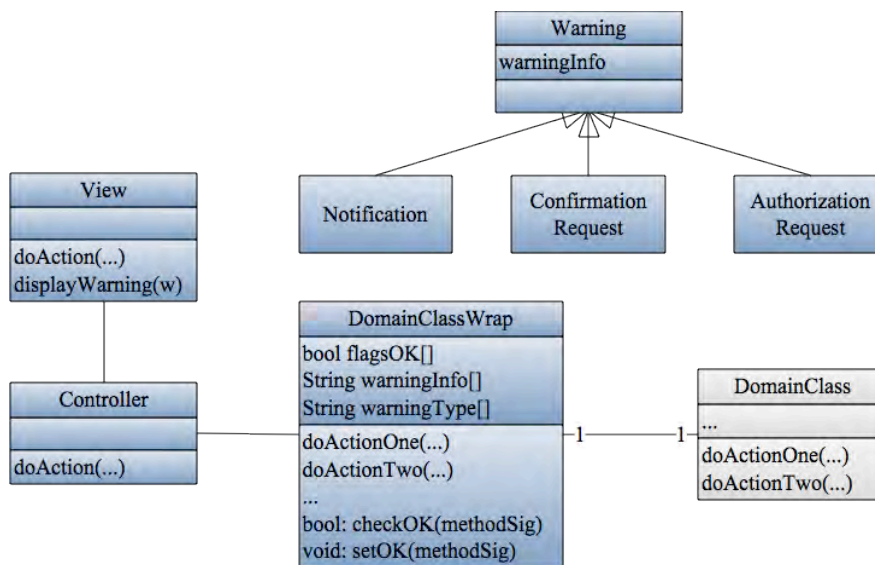


Figure 5.7-3: Usability Design Meta-model. Class diagram. Warning.

#### 5.7.2.3.2 Sequence Diagram “Show Warning”

Figure 5.7-4 shows the sole sequence diagram for this feature, covering all Low-level Design Component Responsibilities described in Table 5.2-6. As explained earlier, a group of one or more System Responsibilities can be represented by a single sequence diagram and vice-versa.

This sequence starts when the user executes an action that may require a warning. This action invocation is captured by the View and forwarded to the Controller as with any other system action. The controller then forwards the call to the domain class it knows (by name) as being responsible for execution of said action.

Any domain class containing ‘sensitive’ methods (those requiring warnings) will be renamed, and a new wrapper class created with its previous name. As explained earlier in the mailing example, if a class like Mailer contains sensitive methods it should be renamed to *MailerPostfix* and a wrapper called Mailer responsible for determining whether or not a warning will be needed for its methods should be created. So, in the sequence diagram shown in Figure 5.7-4, *MailerPostfix* would be represented by DomainClass, and Mailer by DomainClassWrap for this example.

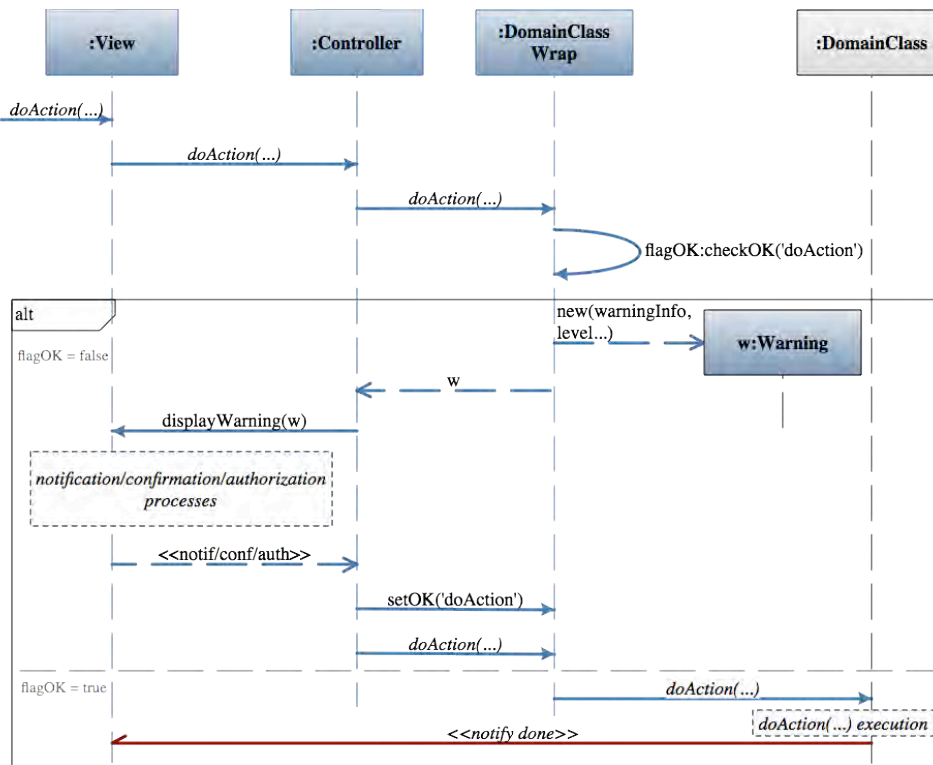


Figure 5.7-4: Sequence Diagram "Show Warning". Warning.

In continuing with the call sequence, once the call reaches the DomainClassWrap, it will check if the method invoked (*doAction()*) is 'ok' to execute. If so (*flagOK = true*, at the bottom of the diagram) DomainClassWrap will simply forward this call to the DomainClass and order it to execute the action as with any other method call. Otherwise (*flagOK = false*) it means the DomainClassWrap must trigger a warning.

DomainClassWrap knows which type of warning is required for each sensitive method of its wrapped class (DomainClass) and what information must go into it. In following with this responsibility, DomainClassWrap creates a Warning object of the appropriate type (Notification, AuthenticationRequest or ConfirmationRequest), fills it with the appropriate information and returns in to the Controller. The controller orders the view to display this Warning object and the view does so.

In any case, after the notification/authorization/authentication has been processed the view forwards the appropriate information resulting from it to the Controller. The controller, in turn, orders the DomainClassWrap to set the OK flag for *doAction()* to TRUE, and orders it to invoke *doAction()* again. At this point DomainClassWrap lets the invocation reach DomainClass, and *doAction()* is ultimately executed.

Classes and methods depicted in dark blue represent they belong to the Warning feature. The notifications in dark red represent Observer Pattern functionality. The gray DomainClass is a template class to be substituted at desing time by the appropriate system class containing the undoable action. For the full color legend see page 71.

## 5.8 “Multi-level Help” Usability Guideline for Software Development

The Multi-level Help Functional Usability Feature covers the user’s need to access the textual help features in different levels of detail throughout a software application. Users may encounter objects within the application and need to know more about them. A need for an expanded help feature explaining the way the application functions is also commonly needed.

The Usability Guideline for Software Development is made up of Analysis and Design artifacts, described for the Warning feature in the following two sections.

### 5.8.1 Usability Guideline for Software Development: Analysis artifacts

There are four artifacts to be used during the analysis phase: the Usability Elicitation Guideline, the Usability Elicitation Clusters, the Usability Use Case Meta-models, the System Responsibilities for Usability. These artifacts are described in the following four sections.

#### 5.8.1.1 Usability Elicitation Guideline

Table 5.8-1 shows the Usability Elicitation Guideline for the Multi-level Help feature. In this guideline, there are three HCI recommendations described below, covering the provision of help as captions and tooltips, as side panels and as a globally accessible feature.

##### 5.8.1.1.1 Captions and Tooltips

HCI authors suggest the simplest level of help that can be provided is the use of captions, prompts and input hints. Also, tooltips are recommended for features that aren’t self explanatory, though their use should not substitute appropriate interface design (MLH\_HCI-1). In the case of tooltips one consideration must be made in the regards to what we have termed ‘dynamic tooltips’. Information shown in a tooltip is usually static, explaining a feature over which the user has placed the mouse. However, tooltips can also be very useful in showing dynamic information not directly encoded in the GUI (MLH\_ELAB-1). The difference may not be obvious initially, but dynamic cases must be pinpointed at elicitation time as they are bound to be more costly to implement and should be estimated accordingly.

Stakeholder discussions should cover which of these three types of help will be made available in the system. Also, which system elements will require or trigger each type of help and with what data (MLH\_Q-1 to MLH\_Q-4). In Table 5.8-1 examples MLH-EX1 and MLH-EX2 describe static vs. dynamic tooltips for this HCI recommendation.

##### 5.8.1.1.2 Help Panels

For items needing to display longer help descriptions, HCI authors suggest setting aside a portion of the page to do so, or to use closeable panels. (MLH\_HCI-2). This type of help is typically dynamic, where information is usually fetched from a repository (U\_ELAB-2). Discussion items for this recommendation involve determining which elements require this type of help, and, for each, what information should be shown to the user (MLH\_Q-5 and MLH\_Q-6). MLH\_EX-3 in Table 5.8-1 describes what is proposed in this recommendation.

##### 5.8.1.1.3 Global Help

For the most elaborate type of help, HCI authors recommend using an external placeholder for the help information, such as a separate page (for web-based applications) or the operating system’s own help software (MLH\_HCI-3). Global help can be accessed in two ways: at its root, when the user is not looking for a specific system feature; this type of help can also be accessed through specific ‘sections’ when the user selects a particular feature of the system and seeks help on it. In this case, the global help will be presented to the user at the specific section, if one exists (MLH\_ELAB-3). Discussions should focus on how this help will be presented and if individual sections will be accessible separately (MLH\_Q-7 to MLH\_Q-9).

Table 5.8-1. Usability Requirements Elicitation Guideline. Multi-level Help.

Identification			
Name	Multi-level help		
Family	Help		
Aliases	Multilevel Help [49]		
Intent			
Providing the user with access to textual help features in different levels of detail			
Problem			
Users may encounter objects within the application and need to know more about them. A need for an expanded help feature explaining the way the application (or parts of it) function is also commonly needed.			
Context			
When the application to be developed is complex and a few users are likely to need a fully-fledged help system, but most users won't take the time to use it; so, developers want to support both impatient and/or occasional users.			
HCI Recommendation	Elaboration	Discussions with Stakeholders	Usage Examples (optional)
<p>MLH_HCI-1 Captions and tooltips</p> <p>Create help on several levels including some (but not all) of the following list. Think of it as a continuum: each of these requires more effort from the user than the previous one:</p> <p>Captions and instructions directly on the page, including patterns like Input Hints and Input Prompt. Be careful not to go overboard with these. If done with brevity, frequent users won't mind them, but don't use entire paragraphs of text – few users will read them.</p> <p>Tooltips. Use these to show brief, one-line descriptions of interface features that aren't self-evident. For icon-only features, these are critical; even nonsensical icons will be taken in stride if the user can tell what it does by rolling over it! Their disadvantages are that they obscure whatever's under them, and that some users find them irritating. A short time delay for the mouse hover – e.g. one or two seconds, removes the irritation factor for most people.</p>	<p>MLH_ELAB-1 Dynamic tooltips</p> <p>Tooltips are often thought of as “static”, meaning that the text they display must always be the same for every instance. However, browser and GUI capabilities as of late also support the use of variables in tooltips. Keep in mind when discussing stakeholders' needs regarding tooltips, in case they seem to be limiting themselves to the static option.</p> <p>Input hints/prompt have no impact on the software architecture</p>	<p>MLH_Q-1 Which of the following types of help will be offered by the system?</p> <p>Tooltips</p> <p>Side Help</p> <p>Global Help</p> <p>MLH_Q-2 If the answer to MLH_Q-1 a) is 'yes', what system elements will require a tooltip to be shown?</p> <p>MLH_Q-3 For each tooltip, what information should be displayed?</p> <p>MLH_Q-4 Of all of these tooltips, which will be dynamic and which will be static?</p>	<p>MLH_EX-1 Web tooltips (static)</p> <p>Placing the cursor over a link within many web applications will display a “tooltip” with, for example, the URL of the page it links to. This type of tooltip is static, and it is encoded directly in the GUI.</p> <p>MLH_EX-2 Dynamic tooltips in Facebook</p> <p>In Facebook, placing the cursor over the Security Level symbol (padlock) of a status update will show a tooltip with the name of the filter that was used when posting said update. This type of tooltip is dynamic, as the system must fetch this value from a repository in order to determine the filter used, as they may be one of many and vary from post to post.</p>
<p>MLH_HCI-2 Help panels</p> <p>Slightly longer descriptions that are shown dynamically as users select or roll over certain interface elements. Set aside an area of the page itself for this, rather than using a tiny tooltip popup.</p> <p>Longer help texts contained inside Closable Panels.</p>	<p>MLH_ELAB-2 Help panels text</p> <p>This type of help text tends to be static but must usually be fetched from memory. Consider that these can serve as triggers to sections of the Global Help if needed.</p>	<p>MLH_Q-5 If the answer to MLH_Q-1 b) is 'yes', what system elements will require side help?</p> <p>MLH_Q-6 For each of these elements, which information should be shown in the side help panel?</p>	<p>MLH_EX-3 Rememberthemilk.com side help</p> <p>In rememberthemilk.com, clicking on a task on the left-hand side of the GUI will display a help box on the right side of the GUI, destined for this purpose.</p>
<p>MLH_HCI-3 Global Help</p> <p>Help shown in a separate window, often done in HTML via browsers, but sometimes in WinHelp or MacHelp. These are often online manuals, entire books, and are reached via menu items on a Help menu, or from “Hlp” buttons on dialogs and HTML pages.</p> <p>“Live” technical support, usually by email, web or telephone.</p>	<p>MLH_ELAB-3 Global Help: Entries</p> <p>The fact that this type of global help can have multiple entry points must be considered: The ‘home’ entry point, when the user is not searching for anything in particular but would like to browse; or any of its sections, if applicable, when the user is searching for something specific.</p>	<p>MLH_Q-7 If the answer to MLH_Q-1 c) how will the Global Help be presented to the user?</p> <p>MLH_Q-8 Will access to individual sections be permitted through specific triggers within the application?</p> <p>MLH_Q-9 If so, which sections will be accessed through which triggers?</p>	<p>MLH_EX-4 Global Help in MS Word</p> <p>In the MS Word task bar, clicking on the “(?) – Help” button will launch the ‘home’ page of the Windows Help Application for MS Word, when working in said OS.</p> <p>MLH_EX-5 Triggers in MS Word Global Help</p> <p>When the user searches in the MS Word Help menu (i.e. looking for a particular term), the menu shows the related topics in real time. Clicking on any of the topics will take the user to that specific section of the Windows Help Application for MS Word.</p>



### 5.8.1.2 Usability Elicitation Clusters

The Usability Elicitation Guideline suggests nine discussion items to be held with stakeholders in order to elicit all aspects of the Multi-level Help Functional Usability Feature. These discussion items can be clearly divided into three initial groups, or clusters, as described in the Usability Elicitation Clusters shown in Figure 5.8-1, according to the portion of the Multi-level Help functionality that they cover.

**MLH\_EC-1 Providing tooltips for individual objects:** The discussion items in this cluster cover determining whether or not tooltips will be required in the system and, if so, for what objects they will be needed and which information should be shown for each.

**MLH\_EC-2 Providing side help for individual objects:** This cluster contains the discussion items related to side help: determining whether or not it will be needed, which objects will trigger it, how the help will be presented graphically and which information should be contained in each.

**MLH\_EC-3 Providing global help at application level:** The discussion items in this cluster cover whether or not there will be global help available from within the application, how it will be displayed for the user and whether or not individual sections of it will be accessible through specific triggers. If individual sections are accessible, the triggers (objects within the application that will require this global help access) need to be specified.

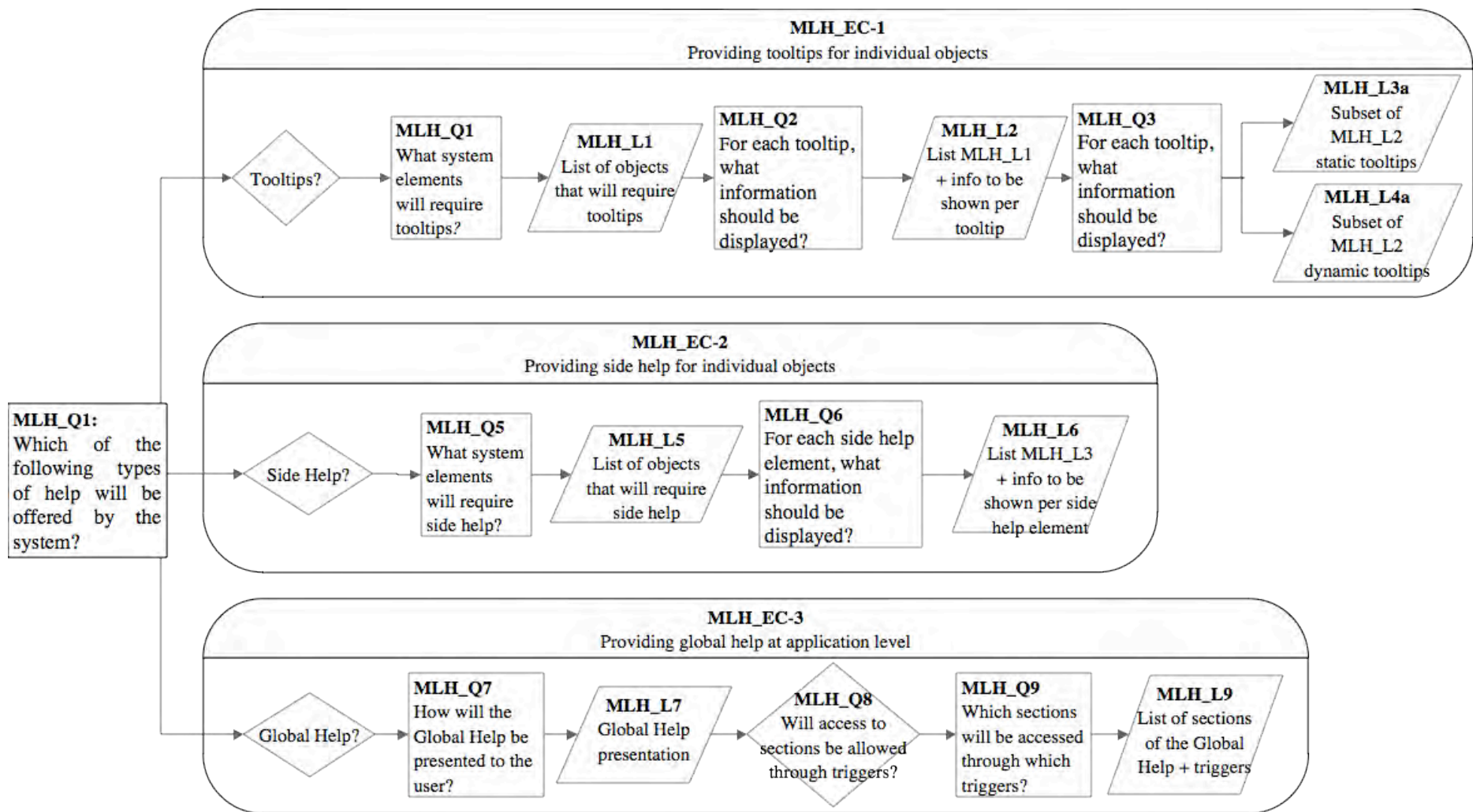


Figure 5.8-1: Elicitation Clusters. Multi-level Help

### 5.8.1.3 Use Case Meta-model

The Use Case Meta-model for the Multi-level Help feature is shown in Figure 5.8-2 (See page 71 for legend), in which five use cases are identified and described below.

**PREF\_UC-1SelectObject:** This template use case represents the selection of an interface object that may result in showing a tooltip (i.e. hovering over a link).

**PREF\_UC-2LoadTooltip:** This use case will always be part of another use case, SelectObject in this case, which will determine whether to show a tooltip.

**PREF\_UC-3LoadSideHelp:** As with LoadTooltip, this use case is invoked from within SelectObject when the selected object requires side help to be loaded.

**PREF\_UC-4LoadGlobalHelp:** This use case starts when the user calls for the Global Help to be shown, without indicating any specific sections to load (the Global Help 'home' will be displayed).

**PREF\_UC-5LoadGlobalHelpSection:** This use case can be started by a user wanting to load a specific section of Global Help. It is also called by the LoadGlobalHelp use case continuously as users move from section to section within it.

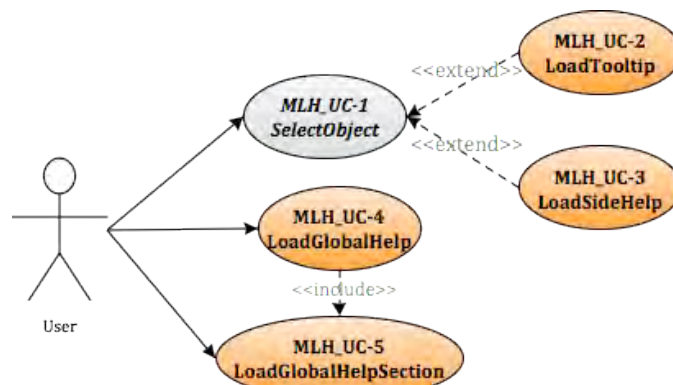


Figure 5.8-2 Use Case Meta-model. Multi-level Help

As mentioned above, the applicability of each of these use cases will depend on the results of the elicitation process. If during elicitation of the Multi-level Help Functional Usability Feature it is determined that tooltips are not needed, for example, the LoadTooltip use case will be ignored. Use cases also depend on one another. These dependencies are shown in Table 5.8-2 where we can see the following:

- The **SelectObject** template use case could potentially (hence the asterisks) need the **LoadTooltip** use case, the **LoadSideHelp** use case or the **LoadGlobalHelpSection** use case. Only one type of help will be required at a time and per selected object.
- The **LoadTooltip** use case needs the **SelectObject** use case to be viable, as tooltips only appear through object selection. The same occurs with **LoadSideHelp**.
- **LoadGlobalHelp** needs no object to be selected for the global help to be accessible, hence the lack of a dependency with SelectObject. However, for GlobalHelp to function, its sections need to be accessible from within it, so it needs the **LoadGlobalHelpSection** use case to be viable.
- **LoadGlobalHelpSection** only needs the template use case **SelectObject** in systems where selecting an object may trigger the loading of the GlobalHelp in a particular section. Otherwise, this use case can be started directly by the user and needs no other use case to be viable, though it's typically present within **LoadGlobalHelp**.

Table 5.8-2 Usability Use Case Dependencies: Multi-level Help Functional Usability Feature

	MLH_UC-1 SelectObject	MLH_UC-2 Load Tooltip	MLH_UC-3 Load SideHelp	MLH_UC-4 Load GobalHelp	MLH_UC-5 LoadGlobal HelpSection
MLH_UC-1 Select Object	-	X*	X*		X*
MLH_UC-2 Load Tooltip	X	-			
MLH_UC-3 Load SideHelp	X		-		
MLH_UC-4 Load GlobalHelp				-	X
MLH_UC-5 LoadGlobalHelpSection	X*				

By looking at the SelectObject row we can see this use case use case is core to the Multi-level Help feature, as expected, since most of the help that is provided is strictly object-dependent. If SelectObject were discarded, however, LoadGlobalHelp would still be viable and, along with LoadGlobalHelpSection, could stand alone.

### 5.8.1.4 System Responsibilities

Table 5.8-3 shows the proposed System Responsibilities for the Multi-level help feature.

Table 5.8-3 System Responsibilities for Multi-level Help

System Responsibilities List for Multi-level Help
MLH_SR-1 Provide tooltips for individual objects The system must know the objects for which it needs to provide tooltips, what information they will display and where it will come from
MLH_SR-2 Provide side help for individual objects The system must know which objects it needs to provide side help for, what information they will display, where this information will come from, and the way in which the side help will be displayed on screen.
MLH_SR-3 Provide Global Help The system must provide access to Global Help from within the application
MLH_SR-4 Provide access to Global Help sections The system must allow access to individual sections of the Global Help when triggered within the application

These System Responsibilities are derived from the Usability Elicitation Clusters as follows:

**MLH\_EC-1 Providing tooltips for individual objects:** This cluster determines waht system objects will require tooltips, the information they will show and whether they will be dynamic or static. As such, this elicitation cluster yields a single System Responsibility, namely **MLH\_SR-1 Provide tooltips for individual objects**.

**MLH\_EC-2 Providing side help for individual objects:** Similarly to the first cluster, this one determines the system objects that will require providing side help, what and how the information will be shown within them, yielding an equivalent System Responsibility: **MLH\_SR-2 Provide side help for individual objects**.

**MLH\_EC-3 Providing global help at application level:** Lastly, this cluster discusses whether access to global help will be needed from within the application, and whether its sections will be accessed individually. If they are, discussions must focus on which objects will link to each section (triggers). This cluster, thus, yields two System Responsibilities: **MLH\_SR-3 Provide Global Help** and **MLH\_SR-4 Provide access to Global Help sections**.

Table 5.8-4 maps the relationships between these Usability Elicitation Clusters and the Usability System Responsibilities, for easy reference. Any project requiring a specific Elicitation Cluster will also need its related System Responsibilities. Likewise, if a cluster is discarded during elicitation, its related responsibilities won't be a part of the resulting system.

Table 5.8-4 Use Case/ System Responsibilities Mapping for Multi-level Help

Use Cases	Dependent Responsibilities
MLH_EC-1 Providing tooltips for individual objects	MLH_SR-1 Provide Tooltips for individual objects
MLH_EC-2 Providing side help for individual objects	MLH_SR-2 Provide Side Help for individual objects
MLH_EC-3 Providing global help at application level	MLH_SR-3 Provide Global Help MLH_SR-4 Provide access to Global Help sections

## 5.8.2 Usability Guideline for Software Development: Design artifacts

The design artifacts of the Usability Guideline for Software Development for this feature are described in the following sections. The System Responsibilities are brought to a lower abstraction level as High-level Design Component Responsibilities in section 5.8.2.1. Section 5.8.2.2 expresses them as Low-level Design Component Responsibilities (for a MVC architecture). Finally, 5.8.2.3 presents the Usability Design Meta-models for said Low-level Design Component Responsibilities as object-oriented class and sequence diagrams.

### 5.8.2.1 High-level Design Component Responsibilities

In order to support the System Responsibilities at design level, the following sections describe the suggested High-level Design Components for the Warning feature, shown in Table 5.8-5.

#### 5.8.2.1.1 User Interface (UI) Component

This component is responsible for listening for all help requests, including mouse hovers over objects that will require tooltips as well as direct invocations of help features. The UI Component is also responsible for relaying information to the user, including appropriate feedback after an action has been executed. Within this feature, such information entails displaying the help information that was requested, appropriately.

#### 5.8.2.1.2 Help Item Component

Help Items hold the actual help information (what will be displayed dynamically in a tooltip, for example) and are associated to specific objects, usually a Domain Component.

#### 5.8.2.1.3 Help Managing Component

This component is responsible for all existing Help Items in the system. The Help Manager Component finds the appropriate Help Item Components upon request and delivers the help information, through the appropriate channels, to the UI Component as needed.

#### 5.8.2.1.4 Domain Component

A Domain Component represents the part of the system to be substituted upon development by the appropriate component. For this feature, this component might be an object (i.e. a bookmark in a web browsing application) that requires a tooltip to be shown whenever the user hovers over its GUI representation (i.e. hovering over a bookmark may display its URL).

Table 5.8-5: Usability Guideline: High-level Design Component Responsibilities. Multi-level Help

S. Responsibility	Generic Component Responsibilities
MLH_SR-1 Provide Tooltips for individual objects	The component responsible for handling interaction with the user (the UI Component) must be aware of calls to activate help over a specific object. Upon reception of such a call, and from a preexisting list, the UI must determine which type of help the object must display In case the type of help to display is a tooltip there are two possible options: a) if the object needs only to display static text, said text will be stored in the UI itself and displayed immediately upon request, and, b) if the text to display is dynamic, the UI must request it from the Help Manager Component (through a delegating component if any) The Help Manager Component is responsible for all existing Help Items. Help Items hold the actual help information--in this case, what will be displayed dynamically in the tooltip--and are associated to a specific object. As such, the Help Manager Component finds the appropriate Help Item Component and returns it to the UI (through a delegating component, if any) The UI then is responsible for displaying the help contents (a tooltip in this case) appropriately for the object over which help was initially invoked.
MLH_SR-2 Provide Side Help for individual objects	In case the type of help to display is of the side-help kind, the UI must request the Help Manager Component (through a delegating component if any) for the text to display. The Help Manager Component must provide the UI with the appropriate Help Item Component (through a delegating component, if any) The UI then is responsible for displaying the help contents (side-help in this case) appropriately for the object over which help was initially invoked.
MLH_SR-3 Provide Global Help	In case the type of help to display is global-help, the UI must request the Help Manager Component (through a delegating component if any) to call unto the OS to open the help feature for this application. The Help Manager Component must provide the OS with the application signature for identification. The OS then carries on the responsibility to display the global help (i.e. WinHelp, external web page, etc)
MLH_SR-4 Provide access to global help sections	If the call to load global-help is accompanied by a specific section identification, the global-help will be loaded, with the exception that the OS must, after loading the global-help, redirect the user to the requested section.

### 5.8.2.2 Low-level Design Component Responsibilities for MVC

When instantiating for a MVC architecture, the **UI** Component is instantiated by View the object(s) and takes over all of its responsibilities except for the delegation of actions to other objects. This responsibility falls on the **Controller** object(s) of the MVC architecture.

The Help Item Component is represented by the extensible **HelpItem** class, and covers all of its responsibilities. The Help Managing component, which is represented by the **HelpManager** class, is a container of HelpItems, and represents the means to access them, as well as the (external) Global Help.

Table 5.8-6 details the Low-level Design Component Responsibilities described above and how they carry out each of the System Responsibilities for Usability. For each System Responsibility, the sequence of actions required by the different objects is presented as well as a set of UML diagrams that depict each of these interactions.

Table 5.8-6: Usability Guideline: Low-level design component responsibilities (MVC). Multi-level Help.

System Responsibility	Objects				Fig
	View	Controller	HelpManager	HelpItem	
MLH_SR-1 Provide Tooltips for individual objects	<ol style="list-style-type: none"> <li>1. The <i>View</i> listens for calls to <code>activateHelp()</code> for a specific object.</li> <li>2. It then determines, from stored static information, what type of help is needed for the object.</li> <li>3. If the type of help needed is a static tooltip, the <i>View</i> displays it immediately. This type of text is stored within the <i>View</i> itself for every object that requires it</li> <li>4. If the type of help needed is a dynamic tooltip, the <i>View</i> asks the <i>Controller</i> to <code>getHelpContents()</code> for the object</li> <li>5. The <i>View</i> displays the help contents according to the type of help it determined the object to need (tooltip).</li> </ol>	<ol style="list-style-type: none"> <li>4. <i>Controller</i> forwards the request to the <i>HelpManager</i></li> <li>7. The <i>Controller</i> forwards the <code>helpContents</code> to the <i>View</i></li> </ol>	<ol style="list-style-type: none"> <li>5. <i>HelpManager</i> determines which <i>HelpItem</i> corresponds to the solicited object and asks it for its contents</li> <li>6. The <i>HelpManager</i> forwards the <code>helpContents</code> to the <i>Controller</i></li> </ol>	<ol style="list-style-type: none"> <li>6. The <i>HelpItem</i> returns its help contents, which in the case of a tooltip is most commonly a short <code>String</code> attribute.</li> </ol>	Figure 5.8-4
MLH_SR-2 Provide Side Help for individual objects	<ol style="list-style-type: none"> <li>1. The <i>View</i> listens for calls to <code>activateHelp()</code> for a specific object.</li> <li>2. It then determines, from stored static information, what type of help is needed for the object.</li> <li>4. If the type of help needed is of the side-help kind, the <i>View</i> asks the <i>Controller</i> to <code>getHelpContents()</code> for the object</li> <li>5. The <i>View</i> displays the help contents according to the type of help it determined the object to need (side-help).</li> </ol>	<ol style="list-style-type: none"> <li>4. <i>Controller</i> forwards the request to the <i>HelpManager</i></li> <li>7. The <i>Controller</i> forwards the <code>helpContents</code> to the <i>View</i></li> </ol>	<ol style="list-style-type: none"> <li>5. <i>HelpManager</i> determines which <i>HelpItem</i> corresponds to the solicited object and asks it for its contents</li> <li>6. The <i>HelpManager</i> forwards the <code>helpContents</code> to the <i>Controller</i></li> </ol>	<ol style="list-style-type: none"> <li>6. The <i>HelpItem</i> returns its help contents, which in the case of a tooltip is likely a series of blocks of structured text</li> </ol>	Figure 5.8-4
MLH_SR-3 Provide Global Help	<ol style="list-style-type: none"> <li>1. The <i>View</i> listens for user calls to <code>loadGlobalHelp()</code> and forwards the request to the <i>Controller</i></li> </ol>	<ol style="list-style-type: none"> <li>2. <i>Controller</i> forwards the request to the <i>HelpManager</i></li> </ol>	<ol style="list-style-type: none"> <li>5. The <i>HelpManager</i> addresses the OS Help System with the request, appending the <code>applicationSignature</code> for identification</li> </ol>		Figure 5.8-5
MLH_SR-4 Provide access to global help sections			<ol style="list-style-type: none"> <li>1. In the case in which <code>loadGlobalHelp()</code> is called with a section, <i>HelpManager</i> must forward this information to the OS Help System as well, to get it to redirect the user to the desired section.</li> </ol>		Figure 5.8-5

### 5.8.2.3 Usability Software Design Meta-models

These UML diagrams represent the Low-level Design Component Responsibilities described in earlier. The following sections describe the class diagram and the classes involved in this feature and their interrelationships, followed by the descriptions of the sequence diagrams.

#### 5.8.2.3.1 Class Diagram

Figure 5.8-3 below shows the class diagram for the Multi-level Help Functional Usability Feature. The main objects involved are the View, Controller, DomainObject, HelpManager, HelpItem, DynamicTooltip, ExtendedHelp. The first two, fulfilling their role within MVC, respectively capture and distribute the user calls to perform actions.

The HelpManager controls all access to any HelpItems available in the system. It is in charge of receiving requests for help from the View through the Controller.

The HelpItem is a parent class, from which concrete classes for every type of help inherit and extend. DynamicTooltip and ExtendedHelp are two such classes. They represent the dynamic tooltip and side help respectively, with the second possibly containing more information (and information *fields*) than the first. Beyond these two classes, however, developers may add more stemming from the parent HelpItem, and have them work in the same way. The DomainObject can be any object for which help (of any type) may be requested.

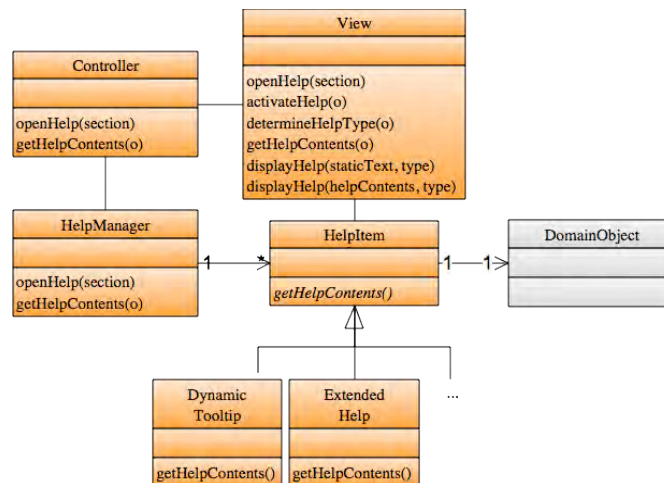


Figure 5.8-3: Usability Design Meta-model. Class diagram. Multi-level Help.

#### 5.8.2.3.2 Sequence Diagram "Get Help"

Figure 5.8-4 shows the sequence diagram MLH\_SR-1 and MLH\_SR-2. A group of one or more System Responsibilities can be represented by a single sequence diagram and vice-versa. This sequence starts when a user 'activates' a request for help over an object (by, for example, hovering over it, or clicking, depending on the item). The View determines the type of help that is being requested and forwards the request to the Controller, save for the case when the help to display is a static tooltip. In such a case, the View directly displays the tooltip, as it possesses all the necessary information to do so (no information needs to be fetched from additional repositories or by performing any calculations, for example, the URL tooltip for a hyperlink).

In all other cases, the call is forwarded to the Controller, who calls on the HelpManager to determine the appropriate HelpItem that is needed for the object in question. Upon finding it, it gathers the textual contents of the Help and passes them back to the View. The view, knowing the type of help that was requested initially, displays these contents accordingly.

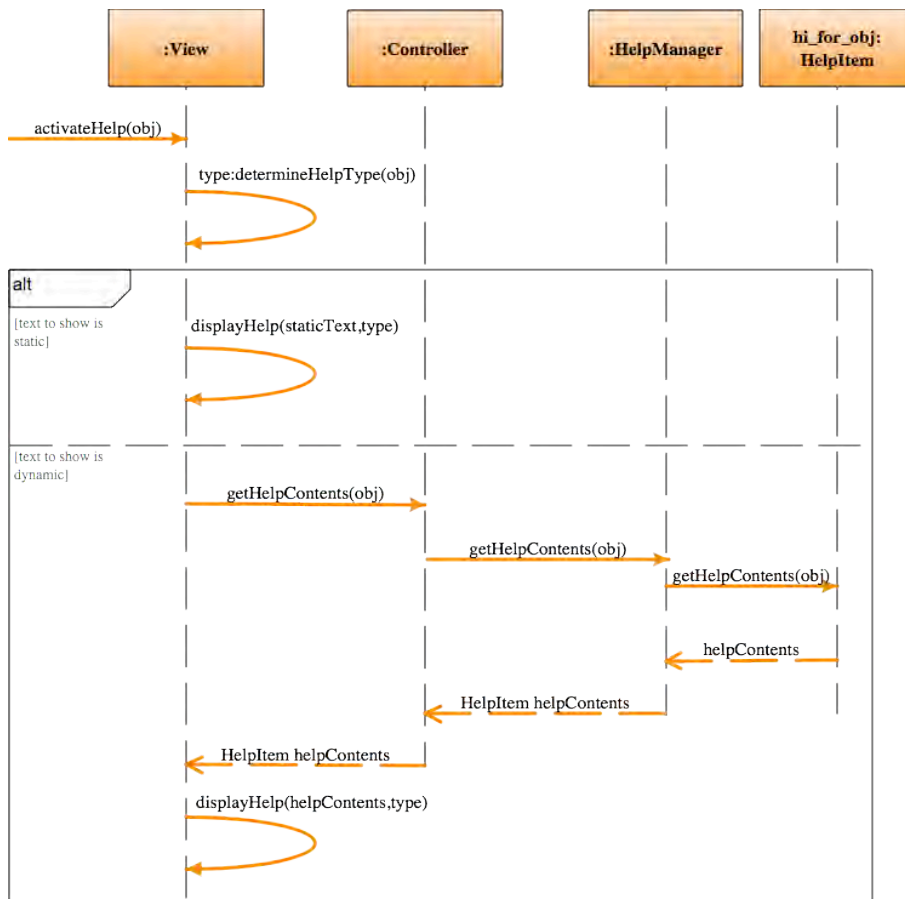


Figure 5.8-4: Sequence Diagram "Get Help". Multi-level Help.

The classes and methods depicted in orange represent that they belong to the Multi-level Help feature. The notifications in dark red represent Observer Pattern functionality. The gray DomainClass is a template class to be substituted at desing time by the appropriate system class containing the action. For the full color legend see page 71.

### 5.8.2.3.3 Sequence Diagram "Get Global Help"

Figure 5.8-5 shows the sequence diagram for getting Global Help This diagram covers all the object responsibilities listed for MLH\_SR-3 and MLH\_SR-4.

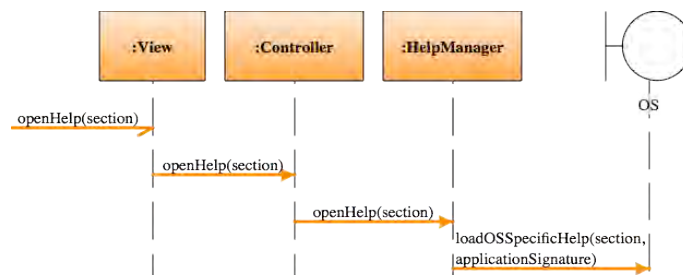


Figure 5.8-5: Sequence Diagram "Get Global Help". Multi-level Help.

The sequence starts off when the user requests a particular section of the global help. The View forwards this call to the controller, who in turn does so to the HelpManager. The HelpManager is responsible for knowing how to locate the global help source outside of the boundaries of the application (typically, the operating system help). In doing so, it calls up the appropriate section in the global help, sending along any additional information, like the application signature, that may berequired by the operating system. From then on the OS takes over, displaying the help that was requested.



## 5.9 “Commands Aggregation” Usability Guideline for Software Development

The Commands Aggregation Functional Usability Feature allows the user to aggregate commands into macro-like structures for ease of batch execution.

The Usability Guideline for Software Development is made up of Analysis and Design artifacts, described for the Commands Aggregation feature in the following two sections.

### 5.9.1 Usability Guideline for Software Development: Analysis artifacts

There are four artifacts to be used during the analysis phase: the Usability Elicitation Guideline, the Usability Elicitation Clusters, the Usability Use Case Meta-models, the System Responsibilities for Usability. These artifacts are described in the following four sections.

#### 5.9.1.1 Usability Elicitation Guideline

Table 5.9-1 shows the Usability Elicitation Guideline for the Commands Aggregation feature. In this guideline, there are six HCI recommendations, described below.

##### 5.9.1.1.1 Macro recording

HCI authors suggest providing a way for the user to record a sequence of actions he may perform repetitively and to save it for future use (CA\_HCI-1). In the cases of more complex macros, the possibility of editing them after creation might arise and must be discussed with stakeholders (CA\_ELAB-1). Stakeholder discussions for this recommendation must determine how recording will take place, which types of actions are suitable for recording, means for stopping a recording session and editing of macros (CA\_Q-1 to CA\_Q-7). In Table 5.9-1, example CA\_EX-1 “Photoshop macro recording” describes an example for this HCI recommendation.

##### 5.9.1.1.2 Macro composition

HCI authors suggest that the user should be allowed to build macros of other macros if desired (nested macros) (CA\_HCI-2). Nested macros tend to be relevant in systems where long sequences of actions are possible, or where very complex actions can occur. (CA\_ELAB-2) Stakeholder discussions must focus on whether or not building these Nested macros or meta-macros will be allowed and, if so, how this will be performed by the user (CA\_Q-8 and CA\_Q-9). Example CA\_EX-2 “Composing Photoshop Macros” in Table 5.9-1 describes an example for this HCI recommendation.

##### 5.9.1.1.3 Macro Play-back

Users should be provided with simple means to play back a previously recorded macro, as per stated in this HCI recommendation (CA\_HCI-3). If a playback feature is to be included in the system, a stop feature should also be considered (CA\_ELAB-3) during elicitation. Stakeholder discussions should focus on determining how the playback functionality will be presented to the user and he will be allowed to stop the macro execution (CA\_Q-10 to U\_Q-12). Example CA\_EX-3 “play/stop buttons” in Table 5.9-1 describes an example for this HCI recommendation.

##### 5.9.1.1.4 Feedback

HCI authors suggest providing the user with appropriate feedback when execution has finished. (CA\_HCI-4). While this is true of most user interactions, completion of a macro should be informed of clearly so the user can move on to perform other actions within the application. (CA\_ELAB-4). The last proposed stakeholder discussion covers the type of feedback to be given to the user upon macro completion (CA\_Q-13). Example CA\_EX-4 “Feedback upon completion” in Table 5.9-1 describes an example for this HCI recommendation.

Table 5.9-1 Usability Requirements Elicitation Guideline. Commands Aggregation.

Identification			
Name	Commands Aggregation		
Family	Commands Aggregation		
Aliases	Composed Command / Macros [42]		
Intent			
Allowing the user to aggregate commands into macro-like structures for ease of batch execution			
Problem			
A user may perform several small tasks during application usage, and some of these may be repetitive. When the user identifies a group of such repetitive actions, establishing a need to perform them more quickly and effortlessly, they will need to aggregate them and invoke them through a single click or call.			
Context			
When the possible actions can be expressed through commands, which can be composed from smaller parts, in a language-like syntax with precise and learnable rules, and the users are willing and able to learn that syntax [42].			
Interrelationships			
Undo: If not considered, none of the macro actions (nor the macro as a whole) will be undoable. Cancel: If not considered, CA_HCI-3Macro play-back cannot be considered in its entirety (a 'stop playback' feature cannot be included). Warning: If not considered, no warnings can be given before macro execution, even for those which may contain damaging actions			
HCI Recommendation	Elaboration	Discussions with Stakeholders	Usage Examples (optional)
CA_HCI-1 Macro recording Provide a way for the user to "record" a sequence of actions. The parts and syntax rules should be easy to learn, and should generate concise commands whose meaning is obvious. The user should be able to give the macro the name of her choice. Let also her to review the action sequence somehow.	CA_ELAB-1 Macro recording: editing and creating Recording a complex macro might result in the need to edit it, be it to fix a mistake in the recording or to add steps. Discuss with stakeholders if macros will be editable.	CA_Q-1 Will user be allowed to record (aggregate) sequences of actions and to save them for later playback? CA_Q-2 Which actions--or types of actions--will be allowed to be recorded? CA_Q-3 How will the user record said actions? CA_Q-4 How will the user stop a recording? CA_Q-5 Will the user be allowed to edit a macro? CA_Q-6 What will be editable in a macro? CA_Q-7 How will the user edit a macro?	CA_EX-1 Photoshop Macro Recording Adobe Photoshop provides a feature for recording user actions for later play back. When the "record" button is clicked every subsequent action performed by the user is recorded, along with any parameters that may have been used (i.e. in "change_color to blue" both the command change_color and the parameter blue are recorded).When the user wishes to stop the recording s/he will click the stop button.The result is a list of all the actions that were taken, where each can be edited by double clicking
CA_HCI-2 Macro Composition Make it possible for one macro to refer to another so they can build on each other [42]	CA_ELAB-2 Macro Composition: Relevance The need for composing macros will usually arise when dealing with complex commands or long sequences, where creating each sequence from scratch might be inconvenient. For systems where long sequences aren't feasible, composing macros is less relevant	CA_Q-8 Will users be allowed to create meta-macros (macros of macros)? CA_Q-9 If so, how will these meta-macros be built?	CA_EX-2 Composing Photoshop Macros Actions recorded to a macro in Adobe Photoshop can be grouped and copy/pasted to another macro, hence composition is allowed through duplication.
CA_HCI-3 Macro play-back Provide a way to the user to "play back" the sequence at any time. The play back should be as easy as giving a single command, pressing a single button, or dragging and dropping an object.	CA_ELAB-3 Macro play-back: stopping A "play back" feature will almost always entail some sort of stop feature. Within a macro, stopping will entail 'cancelling' the current command within the macro being executed and undoing its effects, if any (see Cancel feature)	CA_Q-10How will the user play back a macro? CA_Q-11Will the user be allowed to stop execution of a macro? CA_Q-12If so, how will the execution be stopped?	CA_EX-3 Play/stop buttons Play and Stop buttons are provided within the macro (actions) feature of Adobe Photoshop
CA_HCI-4 Feedback Feedback on the validity of the command or its result should be as immediate as is practical.	CA_ELAB-4 Feedback: Confirmation Like with most other interactions, a confirmatory signal (or message in more complex cases) should be provided.	CA_Q-13What kind of feedback will be provided during recording and/or execution of a macro	CA_EX-4 Feedback upon completion While a macro is executing within Adobe Photoshop the user gets feedback by seeing in real-time the results of each separate action in the macro as it is executed.

### 5.9.1.2 Usability Elicitation Clusters

The Usability Elicitation Guideline suggests thirteen discussion items to be held with stakeholders in order to elicit all aspects of the Commands Aggregation Functional Usability Feature. These discussion items can be clearly divided into five initial groups, or clusters, as described in the Usability Elicitation Cluster Map, shown in Figure 5.9-1, according to the portion of the Commands Aggregation functionality that they cover.

**CA\_EC\_1. Knowing which actions are recordable:** The discussion items in this cluster determine whether macro recording capabilities will be provided within the system and, if so, which actions are suitable for recording.

**CA\_EC\_2. Recording macro:** This cluster covers recording of macros, how it would be done by the user and 'stop recording' capabilities

**CA\_EC\_3. Playing Back Macro:** These discussions deal with how the user will play back a previously recorded macro, whether he'll be allowed to stop playback, and if so, how this will take place.

**CA\_EC\_4. Editing Macro:** This cluster contains the discussions related to macro edition: whether macros will be editable, and, if so, what will be editable in them and how edition will take place.

**CA\_EC\_5. Composing Macros:** The discussion items in this cluster deal with macro composition. If it's allowed within the system, it must be determined how macros will build on one another and how the user will perform such composition.

**CA\_EC\_6. Notifying user of completion:** Stakeholders must determine what type of feedback the user will receive during and/or after macro execution.

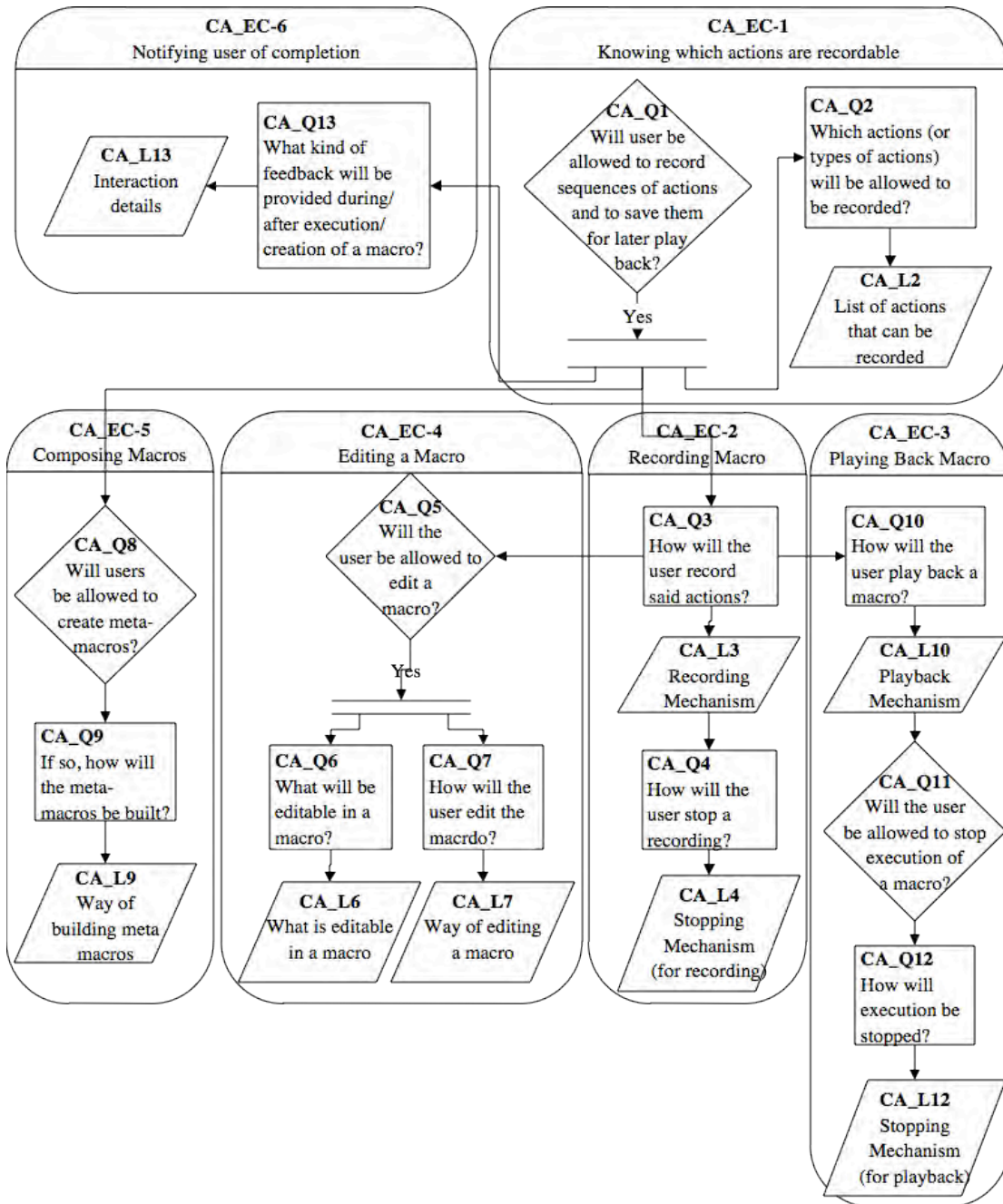


Figure 5.9-1: Elicitation Clusters. Commands Aggregation

### 5.9.1.3 Use Case Meta-model

The Use Case Meta-model for the Commands Aggregation Functional Usability Feature is shown in Figure 5.9-2 (See page 71 for color legend), in which six use cases are identified and described below.

**CA\_UC-1 RecordMacro:** The user selects the option to start recording a macro. Immediately after doing so, the system records every (record suitable) action performed by the user until macro recording is stopped.

**CA\_UC-2 StopMacroRecording:** The user selects the option to stop recording a macro. The system stops recording actions, saves the recorded actions and asks the user to name the newly created macro.

**CA\_UC-3 ComposeMacro:** The user selects the option to compose two or more macros, saving the results to a new macro upon completion.

**CA\_UC-4 EditMacro:** The user selects the option to edit a macro. In doing so, he may compose it with other existing macros. After edition is completed, the user saves the results into the original macro.

**CA\_UC-5 PlayBackMacro:** The user selects an existing macro and orders it to play back. The Macro executes every action that is saved within it (each represented by a different UserAction use case. See below). Once execution is completed, the system informs the user that the macro has finished playing.

**CA\_UC-6 UserAction:** This meta use case represents the actions that make up any system macro.

**A\_UC-1 CancelCommand:** When a macro is playing, the user can select the option to cancel it. Details of how this cancel option is to be implemented should be elicited and detailed as explained in the Abort feature.

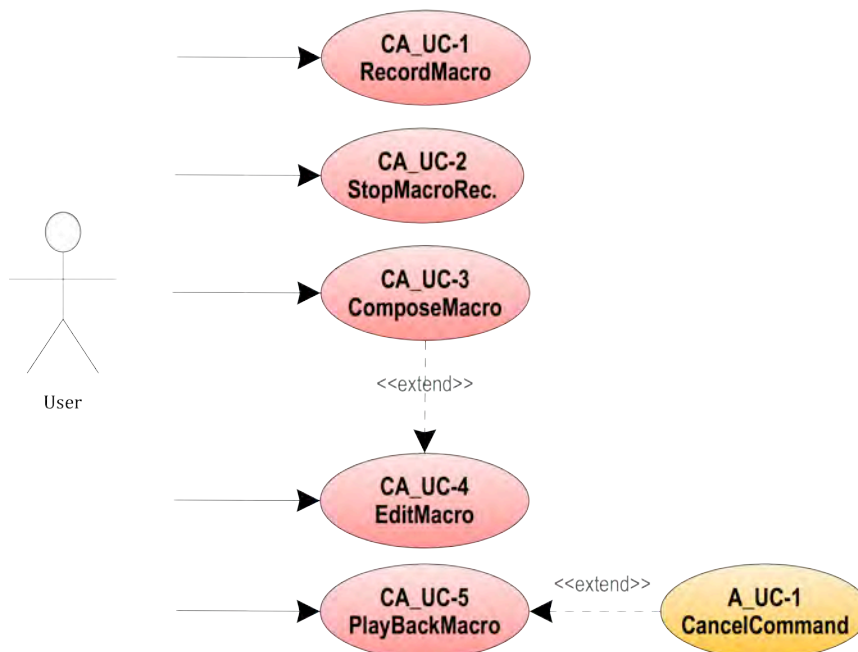


Figure 5.9-2 Use Case Meta-model. Commands Aggregation

The applicability of each of these use cases will depend on the results of the elicitation process. If during elicitation of the Undo feature it is determined, for example, that no Redo feature is needed, then that use case will be discarded. Use cases also depend on one another. These dependencies are shown in Table 5.9-2, where we can see the following:

- The **RecordMacro** use case needs the **UserAction** to be viable, as recordable actions need to exist within the system for recording to be possible
- **Stopping**, composing editing and playing back a macro (CA\_UC-2 to CA\_UC-5) need **RecordMacro** to be viable, as a macro must exist prior to execution of these use cases.
- The **UserAction** template use case needs no other use case to be viable, as it represents any action within the system that is deemed recordable
- The **CancelCommand** use case needs the **UserAction**, as it is the system actions represented by it that will be cancellable

Table 5.9-2 Usability Use Case Dependencies: Commands Aggregation Functional Usability Feature

	CA_UC-1 Record Macro	CA_UC-2 StopMacro Recording	CA_UC-3 Compose Macro	CA_UC-4 Edit Macro	CA_UC-5 PlayBack Macro	CA_UC-6 User Action	A_UC-1 Cancel Command
CA_UC-1 Record Macro	-					X	
CA_UC-2 StopMacroRec.	X	-					
CA_UC-3 ComposeMacro	X		-				
CA_UC-4 EditMacro	X			-			
CA_UC-5 PlayBackMacro	X				-		
CA_UC-6 User Action						-	
A_UC-1 Cancel Command						X	-

By looking at the table columns, it is evident that RecordMacro is core to the Commands Aggregation feature. If it were discarded, no other part of this feature could be implemented.

#### 5.9.1.4 System Responsibilities for Usability

Table 5.9-3 shows the proposed System Responsibilities for Usability for the present feature.

Table 5.9-3 System Responsibilities List for Commands Aggregation

<b>System Responsibilities List for Undo</b>
CA_SR-1 Be aware of which actions are recordable The system must know which actions are recordable
CA_SR-2 Record actions to macro The system must provide the means to record and save macros
CA_SR-3 Playback macro The system must allow for macro playback, providing appropriate feedback after completion
CA_SR-4 Edit macro The system must allow users to edit existing macros
CA_SR-5 Compose macros The system must allow users to compose existing macros, creating new meta-macros

Earlier in this section we identified six elicitation clusters. In the case of most of the other features in this work, a single cluster may yield more than one system responsibility, or, conversely, combining two or more clusters can yield a single system responsibility. In the case of Commands Aggregation there is a one-to-one relationship between clusters and system responsibilities, except for **Notify User of Completion**, which together with **Playing Back Macro** provides **CA\_SR-3 Playback Macro**, as playback and subsequent notification will always happen in sequence. Table 5.9-4 maps these relationships between Usability Elicitation Clusters and the Usability System Responsibilities described above, for reference.

Table 5.9-4 Use Case/ System Responsibilities Mapping for Commands Aggregation

Use Cases	Dependent Responsibilities
CA_EC-1 Knowing which actions are recordable	CA_SR-1 Be aware of which actions are recordable
CA_EC-2 Recording Macro	CA_SR-2 Record actions to macro
CA_EC-3 Playing Back Macro	CA_SR-3 Playback macro
CA_EC-4 Editing Macro	CA_SR-4 Edit macro
CA_EC-5 Composing Macros	CA_SR-5 Compose macros
CA_UC-6 Notifying user of completion	CA_SR-3 Playback macro

## 5.9.2 Usability Guideline for Software Development: Design artifacts

The design artifacts for the Commands Aggregation feature are described below. In section 5.9.2.1 the System Responsibilities are brought to a lower abstraction level as High-level Design Component Responsibilities, and in section 5.9.2.2 as Low-level Design Component Responsibilities for MVC. Finally, section 5.9.2.3 shows the Usability Design Meta-models.

### 5.9.2.1 High-level Design Component Responsibilities

Widely known research results have already addressed part of the expected functionality of this feature. Such is the case of the Command Pattern by the GoF, which we have chosen as the core of this feature. Components (and, in the following sections, classes and objects) like the “Command” component are taken directly from this widely known design pattern to fulfill the needs that arise from the corresponding System Responsibilities.

A smaller portion of the High-level Design Component Responsibilities described in the following subsections, like some contained in the Macro Component, are not addressed by the original GoF Command pattern, and so are included as part of the original contribution of this work in order to fulfill the entirety of the expectations of this feature. Another pattern used across all the guidelines is the GoF Observer pattern, a defining part of the MVC architecture itself. All of these components are described below and summarized in Table 5.9-5.

#### 5.9.2.1.1 User Interface (UI) Component

The UI is responsible for capturing all user invocations and forwarding them (possibly through a delegating component) to the appropriate part of the domain, usually that responsible for executing the action. The UI Component is also responsible for relaying information to the user, including appropriate feedback after an action has been executed.

#### 5.9.2.1.2 Domain Component

A Domain Component represents the part of the system that is ultimately responsible for executing the actions requested by the user.

#### 5.9.2.1.3 Command Component

Based on Gamma’s definition of the Command Pattern, the Command Component is responsible for encapsulating method invocations and any pertinent state information at call-time. When an action is invoked through the UI Component, it would normally be forwarded directly to the Domain Component responsible for executing said action. However, when the action that is called needs to be stored when recording, additional steps need to be taken before the Domain Component is allowed to execute it.

After the call is placed in the UI, a new instance of Command is created and initialized with the signature of the method being called and a reference to the Domain Component in charge of executing it, for later invocation. It is also initialized with any state information that will need to be restored when called again from a macro. Furthermore, the Command is responsible for calling the method in the Domain Component (after saving the state).

#### 5.9.2.1.4 Recording Component

This component is responsible for recording a macro (ideally in a separate execution thread). It stores all recordable actions that are performed by the user, until it’s ordered to stop.

#### 5.9.2.1.5 Macro Component

The Macro Component represents a sequence of previously recorded actions. It’s a collection of instances of the Command component and, when ordered, this component is responsible for invoking all the commands that compose it, in the appropriate order.

Table 5.9-5: Usability Guideline: High-level Design Component Responsibilities. Commands Aggregation

System Responsibility	Generic Component Responsibilities
CA_SR-1 Be aware of which actions are recordable	During macro recording (see 0), the component responsible for invoking commands ( <i>Invoker</i> ) must determine if an action that is performed by the user is a 'recordable' action (one that can be added to a macro)
CA_SR-2 Record actions to macro	<p>The component in charge of processing user input (<i>UI Component</i>) is responsible for listening for user calls to start recording a new macro</p> <p>The component in charge of delegating actions (if any) must forward this request to a <i>Recording Component</i></p> <p>The <i>Recording Component</i> is responsible for recording to a new macro (preferably in a separate thread) all recordable actions performed by the user, until its ordered to stop.</p> <p>A macro is a <i>ConcreteCommand</i>. A regular <i>Concrete Command Component</i> encapsulates a call to an action in a <i>DomainClass</i> component, the parameters it's called with, state information, and other relevant data. This encapsulation allows for <i>Commands</i> to be recreated further down the road. Macros are designated as a special kind of <i>ConcreteCommand</i> (one without a unique association to a <i>DomainClass</i>) so that they can be composed.</p> <p>Once an empty macro has been created, and while recording is active, the <i>UI</i> (through any delegating component) must continue to listen to actions invoked by the user.</p> <p>When a new action is invoked, the delegating component (if any) orders the action to execute. It does so by instantiating a <i>Command Component</i> for the called action, adding it to the macro, and then ordering said <i>Command</i> to execute. The delegating component will continue to treat new actions in this manner (adding them to the macro) until it is ordered to stop.</p> <p>The <i>UI Component</i> continually listens for a 'stop' order from the user. When it receives one it forwards it to the delegating component (if any), which orders the <i>Recorder Component</i> to stop recording, thus completing the newly created macro.</p>
CA_SR-3 Playback macro	<p>The <i>UI Component</i> is responsible for listening for user calls to play back a specific macro</p> <p>The delegating component (if any) is responsible for locating called macros and ordering them to execute</p> <p>Each macro is represented within the application by a <i>Macro Component</i>. When called, this component is responsible for invoking all the commands that compose it, in the appropriate order</p>
CA_SR-4 Edit macro	<p>The <i>UI Component</i> is responsible for listening for user calls to edit a specific macro</p> <p>The delegating component (if any) is responsible for locating the corresponding <i>Macro Component</i> and sending it to the <i>UI</i> for editing</p> <p>The <i>UI</i> is also responsible for displaying the editable fields to the user, capturing any modifications and for sending them back (through a delegating component, if any) to the <i>Macro Component</i> for saving</p> <p>The <i>Macro Component</i> is responsible for saving those edits and modifying any of the commands that compose it (when/if affected)</p>
CA_SR-5 Compose Macros	<p>To compose two macros, a 'target' macro has the capability to append another macro at a specified position within its internal list of commands</p> <p>The <i>UI Component</i> is responsible for listening for user calls to compose two existing macros (a 'target macro' and a 'macro to append').</p> <p>The delegating component (if any) is responsible for locating both macro components and for instructing the 'target macro' to insert, at the specified position within its command list, the 'macro to append'.</p>

### 5.9.2.2 Low-level Design Component Responsibilities for MVC

When instantiating for a MVC architecture, the **UI Component** is instantiated by **View** the object(s) and takes over all of its responsibilities except for the delegation of actions to other objects. This responsibility falls on the **Controller** object(s) of the MVC architecture.

Likewise, the **Command Component** is defined in the **Command** interface and implemented by **ConcreteCommand** objects. For every recordable command there will be a distinct **ConcreteCommand** class (i.e. **ExportVideoCommand**, **OpenDoorCommand**, etc.). Whenever a command is called through the **View**, the corresponding **ConcreteCommand** object will be created, saved to history (so that it can be called in the future from within a macro, with the original parameters and state), and only then ordered to execute.

The **Macro component** is represented by the **MacroCommand**, which is also defined in the **Command** interface, but, additionally, may contain a list of other **Commands** (and, by definition, other **MacroCommands**, allowing for nesting of macros). The **Recorder component** is represented by **Recorder**, which orders macro recording and stopping as requested by the **View**. Finally the **Domain Component** is represented by the **DomainClass**, and contains the actual actions to execute when recording and playing macros.

Table 5.9-6 details the Low-level Design Component Responsibilities described above and how they carry out each of the System Responsibilities. For each System Responsibility, the sequence of actions required by the different objects is presented as well as a set of UML diagrams that depict each of these interactions.



Table 5.9-6: Usability Guideline: Low-level Design Component Responsibilities (MVC). Commands Aggregation.

System Responsibility	Objects					Fig
	View	Controller	Recorder	ConcreteCommand	DomainClass	
CA_SR-1 Be aware of which actions are recordable		1. When 'recording' is active, the <i>Controller</i> is responsible for knowing whether a called method, <code>doAction()</code> , is recordable or not.				Figure 5.9-4
CA_SR-2 Record actions to macro	1. The <i>View</i> listens for the user call to <code>startRecording(...)</code> . To do so, the user enters the name s/he wishes to give to the new macro and any other pertinent information. 2. The <i>View</i> forwards this call to the <i>Controller</i> 6. When the user invokes a new action, <code>doAction()</code> , the <i>View</i> forwards it to the <i>Controller</i> 10. When the user orders to <code>stopRecording()</code> , the <i>View</i> forwards the call to the <i>Controller</i>	2. The <i>Controller</i> asks the <i>Recorder</i> to start recording by invoking its <code>record()</code> method. 7. The <i>Controller</i> then determines, from a preexisting list, for example, if <code>doAction()</code> is listed as 'recordable'. If it's not, it simply <code>executes()</code> it. Otherwise it instantiates a <i>ConcreteCommand</i> to represent it, <code>clones()</code> it and <code>adds()</code> it to the macro. 11. When ordered to <code>stopRecording()</code> , the <i>Controller</i> forwards the call to the <i>Recorder</i> .	3. When ordered to start recording, the <i>Recorder</i> sets a flag to <code>TRUE</code> to indicate that recording is 'active' 4. It then creates a new (empty) <i>ConcreteCommand</i> object, representing the new (yet empty) macro. 5. The <i>Recorder</i> then waits for the next recordable action to be invoked 12. When ordered to <code>stopRecording()</code> , the <i>Recorder</i> sets the 'recording' flag to <code>FALSE</code> and effectively stops adding commands to the macro.	8. When the <i>ConcreteCommand</i> is ordered to <code>execute()</code> , it calls upon its <i>DomainClass</i> and orders the initial, <code>doAction()</code> , method to be executed.	9. When <i>DomainClass</i> receives the order, it <code>executes doAction()</code>	Figure 5.9-4
CA_SR-3 Playback macro	1. The <i>View</i> listens for the user call to <code>executeMacro(macroId)</code> and forwards it to the <i>Controller</i> .	2. The <i>Controller</i> forwards the call to <code>executeMacro(macroId)</code> , and then orders the <i>ConcreteCommand</i> object that represents that macro to <code>execute()</code>		3. When ordered to <code>execute()</code> the <i>ConcreteCommand</i> (macro) invokes the <code>execute()</code> method of all of its internal <i>ConcreteCommands</i> , which in turn order their <i>DomainClasses</i> to execute the appropriate action	4. Each <i>DomainClass</i> will execute the called action.	Figure 5.9-5
CA_SR-4 Edit macro	1. The <i>View</i> listens for the user call to <code>viewMacro(macroId)</code> and forwards it to the <i>Controller</i> . 2. When it receives a <i>ConcreteCommand</i> object (macro) it presents the user with editing fields corresponding to every part of the macro (and its internal <i>Commands</i> if applicable) that is editable. 3. The user then edits any/all the information and the <i>View</i> passes on these edits to the <i>Controller</i> through <code>saveMacroEdits(...)</code>	2. The <i>Controller</i> finds the appropriate <i>ConcreteCommand</i> object (macro) and returns it to the <i>View</i> 3. After receiving the edits, the <i>Controller</i> orders the <i>ConcreteCommand</i> (macro) to <code>modify()</code> itself with these new <code>edits[]</code>		3. When ordered to <code>modify()</code> , the <i>ConcreteCommand</i> (macro) takes the <code>edits[]</code> and applies them, <code>modifying()</code> any internal <i>ConcreteCommands</i> if needed.		Figure 5.9-6
CA_SR-5 Compose Macros	1. The <i>View</i> listens for the user call to <code>composeMacro(macro1, macro2)</code> and forwards it to the <i>Controller</i> .	2. The <i>Controller</i> finds the appropriate <i>ConcreteCommand</i> object (macro2) and orders it to <code>insert()</code> macro1 and the position <code>pos</code>		3. The <i>ConcreteCommand</i> <code>macro2</code> is now composed of the original <code>macro2</code> and <code>macro1</code> appended at <code>pos</code>		Figure 5.9-7

### 5.9.2.3 Usability Software Design Meta-models

This section describes the UML diagrams representing the Low-level Design Component Responsibilities. Below, the class diagram for this feature is presented, along with a description of the classes and interrelationships involved, as well as the sequence diagrams.

#### 5.9.2.3.1 Class Diagram

Figure 5.9-3 below shows the class diagram for the Commands Aggregation Functional Usability Feature. The main objects involved are the View, Controller, Command, ConcreteCommand, Recorder, DomainClass and MacroCommand. The first two, fulfilling their role within MVC, respectively capture and distribute the user calls to perform actions.

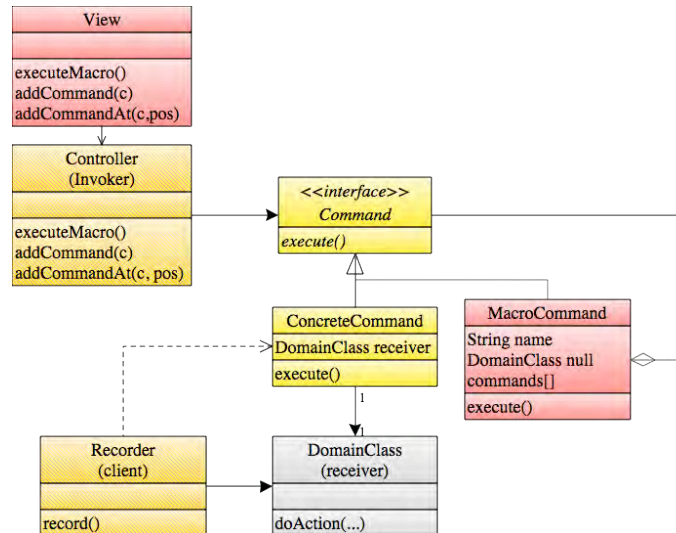


Figure 5.9-3: Usability Design Meta-model. Class diagram. Commands Aggregation.

The ConcreteCommand class implements a Command interface, as described by GoF's Command Pattern, and is responsible for ordering the execution of the requested action (in DomainClass) as well as for storing all necessary state information required for eventually replaying the command it represents (i.e. the method it is calling within DomainClass). In any given system there will be as many ConcreteCommands as there are undoable actions, and it is recommended they be labeled by an appropriate mnemonic. For example, the ConcreteCommand class in charge of invoking the sendMail() method in an email application should be labeled something like SendMailCommand or SendMailConcreteCommand, etc.

The Recorder class represents the client within the GoF's Command Pattern ordering the starting and stopping of macro recording and the MacroCommand holds a list of its commands (or macro commands) for future execution.

These classes and interfaces which belong to the Command Pattern are depicted in yellow to differentiate them from the rest of this feature's classes. See appendix [C] for color legend.

#### 5.9.2.3.2 Sequence Diagram "Record Macro"

Figure 5.9-4 shows the sequence diagram for CA\_SR-1 Knowing which actions are recordable and CA\_SR-2 Record actions to macro.

This sequence starts when the user orders to record a new macro. It may provide a name for it along with any other optional parameters. The View captures this call and forwards it to the Controller, which orders the Recorder to start recording. The Recorder sets its 'recording' flag to TRUE, creates a new instance of MacroCommand and listens for every new recordable action that may be invoked to store it in this new macro.

After recording starts, every new action that is invoked by the user goes through the steps suggested by the GoF in their Command pattern: prior to execution, the ConcreteCommand which encapsulates the invoked action (doAction()) is cloned (preserving its state information intact) and stored, in this case in the HistoryList object. After that, it is ordered to execute(), which entails a call to the desired method of the corresponding DomainClass. When the user orders the system to stop recording this call is captured by the View, and forwarded through the Controller to the Recorder, which sets its recording flag to FALSE and no longer listens for invoked actions from the user.

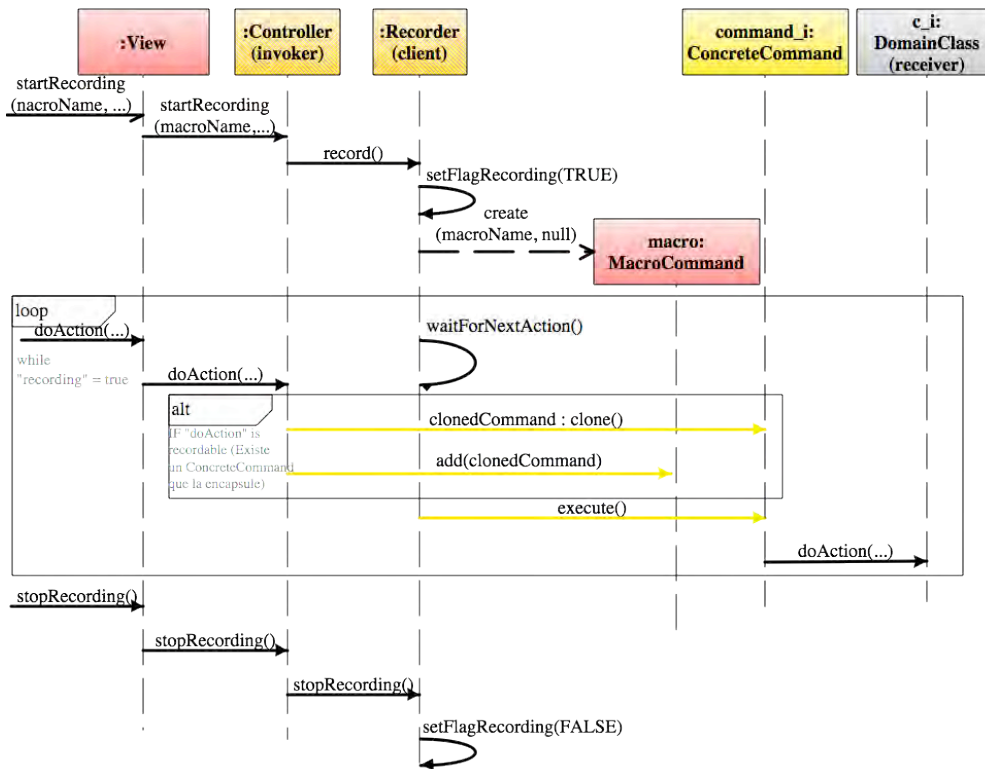


Figure 5.9-4: Sequence Diagram "Record Macro". Commands Aggregation.

Classes and methods depicted in salmon color represent they belong to the Commands Aggregation feature. Those in yellow are part of the Command, those in a mix of salmon and yello perform actions pertinent to both the Command pattern and our original contribution, and notifications in dark red represent Observer Pattern functionality. The gray DomainClass is a template class to be substituted at desing time by the appropriate system class containing the undoable action. For the full color legend see page 71.

### 5.9.2.3.3 Sequence Diagram "Execute Macro"

Figure 5.9-5 shows the sequence diagram that covers the CA\_SR-3 Playback Macro system responsibility as well as all of its corresponding low-level responsibilities.

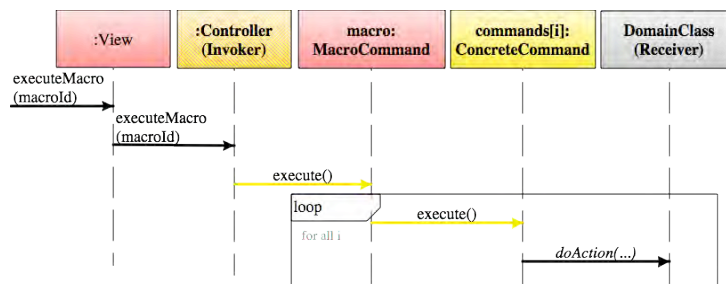


Figure 5.9-5: Sequence Diagram "Execute Macro". Commands Aggregation.

The sequence starts off when the user requests to execute a particular macro, providing the View with its identification information. The View forwards this call to the Controller which locates the MacroCommand being requested and orders it to execute.

When a MacroCommand is ordered to execute, it calls every ConcreteCommand within it and orders it to execute in turn.

#### 5.9.2.3.4 Sequence Diagram "Edit Macro"

Figure 5.9-6 shows the sequence diagram that covers CA\_SR-4 Edit Macro. The sequence starts when the user opens an existing macro. The Controller locates the macro, forwards it to the View and the View displays it for the user for editing.

Since editing of macros will vary from system to system depending on the domain, the specific methods pertaining to the macro edition will need to be filled in at design time. Once the macro has been edited, the View forwards the order to save it, along with the changes, to the Controller. The controller passes on these changes to the MacroCommand object, which incorporates them within it and all of its ConcreteCommands that may be affected.

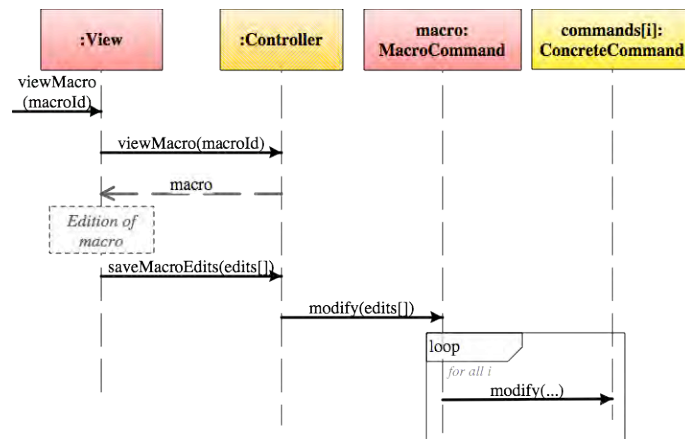


Figure 5.9-6: Sequence Diagram "Edit Macro". Commands Aggregation.

#### 5.9.2.3.5 Sequence Diagram "Compose Macro"

Figure 5.9-7 shows the sequence diagram that covers the CA\_SR-5 Compose Macros system responsibility as well as all of its corresponding Low-level Design Component Responsibilities.

This simple sequence shows how the call to compose two macros is forwarded from the View to the Controller, which in turn finds the second macro in the sequence and orders it to insert all commands found in the first macro at its head. This results in this second macro now containing all the actions of the first macro, followed by its own.

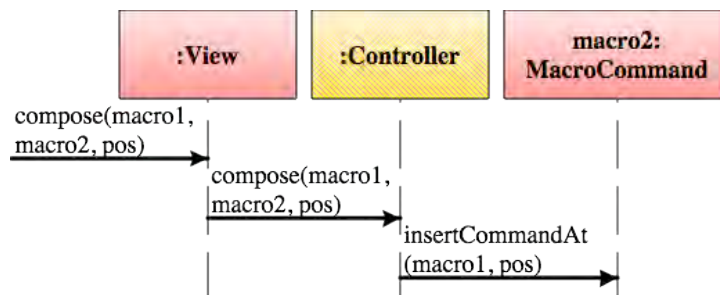


Figure 5.9-7 Sequence Diagram "Compose Macro". Commands Aggregation

## 5.10 “Preferences” Usability Guideline for Software Development

The Preferences Functional Usability Feature covers the user’s need to for a centralized place where they can alter the application’s settings. The main issue that this feature tries to address is the fact that an application may have multiple ways in which it can function, depending on each user’s needs and tastes. Thus, there is a need to allow individual users to chose among them, and to let them save such configurations as their own for future use.

The Usability Guideline for Software Development is made up of Analysis and Design artifacts, described for the Commands Aggregation feature in the following two sections.

### 5.10.1 Usability Guideline for Software Development: Analysis artifacts

There are four artifacts to be used during the analysis phase: the Usability Elicitation Guideline, the Usability Elicitation Clusters, the Usability Use Case Meta-models, the System Responsibilities for Usability. These artifacts are described in the following four sections.

#### 5.10.1.1 Usability Elicitation Guideline

Table 5.10-1 shows the Usability Elicitation Guideline for the Preferences Functional Usability Feature. In this guideline, there are three HCI recommendations, explained below.

##### 5.10.1.1.1 Common Usage of Preferences

HCI authors suggest that application users be provided with a space where they can change (and save) application values, from simple color schemes to settings for language support (PREF\_HCI-1). It is important to note that regardless of the settings being altered, this HCI recommendation contemplates only their selection, organization, storage, etc. It does not delve into what use the application will give to each setting or what they will mean within the application domain (PREF\_ELAB-1). Stakeholder discussions PREF\_Q-1 to PREF\_Q-3 elicit the needs regarding which preferences will be made available for the system and how they’ll be handled for different users, if applicable. Table 5.10-1 shows an example for this HCI recommendation in example PREF-EX-1 “Gmail Preferences”.

##### 5.10.1.1.2 Default Settings

In most cases, a preference will have a default value, or one that is automatically chosen before the user makes their own selection (or chooses not to).

When applications have a large number of preferences that can be set by the user, HCI authors recommend providing ‘canned settings’ or groups of preferences with preset default values, that the user can choose from, instead of picking individual values. (PREF\_HCI-2). These canned settings can range from a single ‘default setting’ to multiple canned settings, depending on the amount of individual preferences available to the user, the frequency of change, etc. all of which should be evaluated during elicitation (PREF\_ELAB-2). Default values for individual preferences and canned settings, as well as the number of canned settings to provide should be thoroughly discussed (PREF\_Q-4 to PREF\_Q-6). PREF-EX-2 “Gmail Default” in Table 5.10-1 describes an example for this HCI recommendation.

##### 5.10.1.1.3 Grouping Preferences

PREF\_HCI-3 describes how similar preferences can be grouped together when dealing with a large sets. This grouping can be done linearly or forming a tree structure. Either of these add complexity to the setting of preferences and should be carefully decided upon during elicitation of this feature (PREF\_ELAB-3). Stakeholders shall discuss whether either of these structures is required, or if preferences will be handled individually (PREF\_Q-7 and PREF\_Q-8). Table 5.10-1 describes an example for this HCI recommendation when using a tree structure, in example PREF-EX-2 “Eclipse Preferences”.

Table 5.10-1. Usability Requirements Elicitation Guideline. Preferences.

Identification			
Name	Preferences		
Family	User Profile		
Aliases	Preferences [49]; User preferences [42]		
Intent			
The Preferences feature provides users with a centralized place where they can alter the application's settings.			
Problem			
An application may have multiple ways in which it can function, depending on each user's needs and tastes. There is a need to allow individual users to chose among them, and to let them save such configurations as their own for future use.			
Context			
When the application is very complex and many of its functions can be tuned to the user's preference, and/or the system will be used by people with different abilities, cultures and tastes, and not enough is known about the user's preferences in order to assume defaults that will suit all users.			
Interrelationships			
When including the Preferences feature in an application, in order to allow users to discard changes made to their preferences, Undo must be considered. Also if loading a set of preferences takes more than a few seconds the Progress feature will be needed			
HCI Recommendation	Elaboration	Discussions with Stakeholders	Usage Examples (optional)
<p>PREF_HCI-1 Common usage of preferences</p> <p>Provide a place or working surface where users can pick their own settings for things like language, fonts, icons, color schemes, and use of sound. Allow users to save those preferences, so that they do not have to spend time setting them again, but do this per user if multiple people will use it [42]. Let those preferences become the default for each user on further use, [49].</p>	<p>PREF_ELAB-1 Common usage of prefs.</p> <p>These recommendations extend to the organization and setting of preferences only. What the system will do or how it will change for each falls outside of the scope of this feature. For example, changing the Language preferences of an application would be considered as part of the Preferences feature, but the actual display of the application text in the different languages would not (only the selection, organization and storage of language possibilities and the user interaction required to change them).</p>	<p>PREF_Q-1 Will users be allowed to set up preferences?</p> <p>PREF_Q-2 If so, which preferences will users be allowed to set up?</p> <p>PREF_Q-3 Will preferences be global to the application or particular to each user?</p>	<p>PREF_EX-1 Gmail preferences</p> <p>Gmail provides the user with a section devoted to changing preferences like number of emails shown per page, alt. Address, away auto-response, and many more. It also allows for setting the visual aspects of the application, like the background and foreground colors and multiple images found throughout the UI.</p>
<p>PREF_HCI-2 Default settings</p> <p>Devise a set of alternative "canned settings" that users can choose between, if they don't like the default and don't want to spend hours picking out good combinations [42].</p>	<p>PREF_ELAB-2 Default Settings</p> <p>Discuss with the user which of these canned settings will make sense. Sometimes only one of them will be enough, but others an array of canned settings might be preferable</p>	<p>PREF_Q-4 What will be the default values for the preferences listed in PREF_Q-2?</p> <p>PREF_Q-5 Aside from the default values, will the system provide groups of "canned settings" for the user to choose from?</p> <p>PREF_Q-6 If so, what will these groups of settings be?</p>	<p>PREF_EX-2 Gmail defaults</p> <p>Gmail preferences can always be reverted to one default setting (i.e. factory setting). As for the look and feel of the application, there are dozens of canned settings or 'themes' to chose from.</p>
<p>PREF_HCI-3 Grouping preferences</p> <p>If the number of groups is small, property pages can be used for each group but when the number of groups is high, use a tree [49].</p>	<p>PREF_ELAB-3 Grouping preferences</p> <p>Preferences should be grouped in a way that makes sense to the user. Preferences in the same group, whether property pages or a tree, must be of the same "type" or affect the same parts of the system.</p>	<p>PREF_Q-7 Will preferences be arranged in groups or in a tree structure instead of handled individually?</p> <p>PREF_Q-8 If groups or trees are to be used, what will be their structure?</p>	<p>PREF_EX-3 Gmail vs. Eclipse</p> <p>Gmail stores its preferences in property pages, while Eclipse's high number of preferences is arranged in a tree structure.</p>

### 5.10.1.2 Usability Elicitation Clusters

The Usability Elicitation Guideline suggests eight discussions items (PREF\_Q-1 to PREF\_Q-8) to be held with stakeholders in order to elicit their needs in respect to the Preferences Functional Usability Feature. These discussion items can be divided into five clusters (as described in the Usability Elicitation Cluster shown in Figure 5.10-1), according to the portion of the Preferences functionality that they cover.

**PREF\_EC-1 Providing Preferences:** The discussion items in this cluster determine if setting preferences will be needed at all within the application and, if so, which preferences the users will be allowed to set up and save.

**PREF\_EC-2 User-specific vs. Global Preferences:** If preferences are to be provided, the discussion items in this cluster determine whether they will be user-specific, meaning that each user will be able to modify and save their own preferences independently, or global for the entire application, regardless of who is using it.

**PREF\_EC-3 Default Values:** The discussion item in this cluster determines the desired default value for each preference addressed in the discussions of elicitation cluster PREF\_EC-1 Providing Preferences. The default value is intended to be the preference's value when/if users have not assigned it a value themselves or do not intend to do so.

**PREF\_EC-4 Canned Settings:** This elicitation cluster determines whether canned settings, or groups of pre-determined values for all preferences for the user to choose from, will be provided.

**PREF\_EC-5 Grouping Preferences:** Finally, this last elicitation cluster contains the discussion items regarding the way in which preferences will be presented to the user. They can be all presented together ('no' branch in Figure 5.10-1) or using some kind of structure ('yes' branch in Figure 5.10-1). The structures to use can be groups or trees, and stakeholders must decide which, if either, in these discussion items.

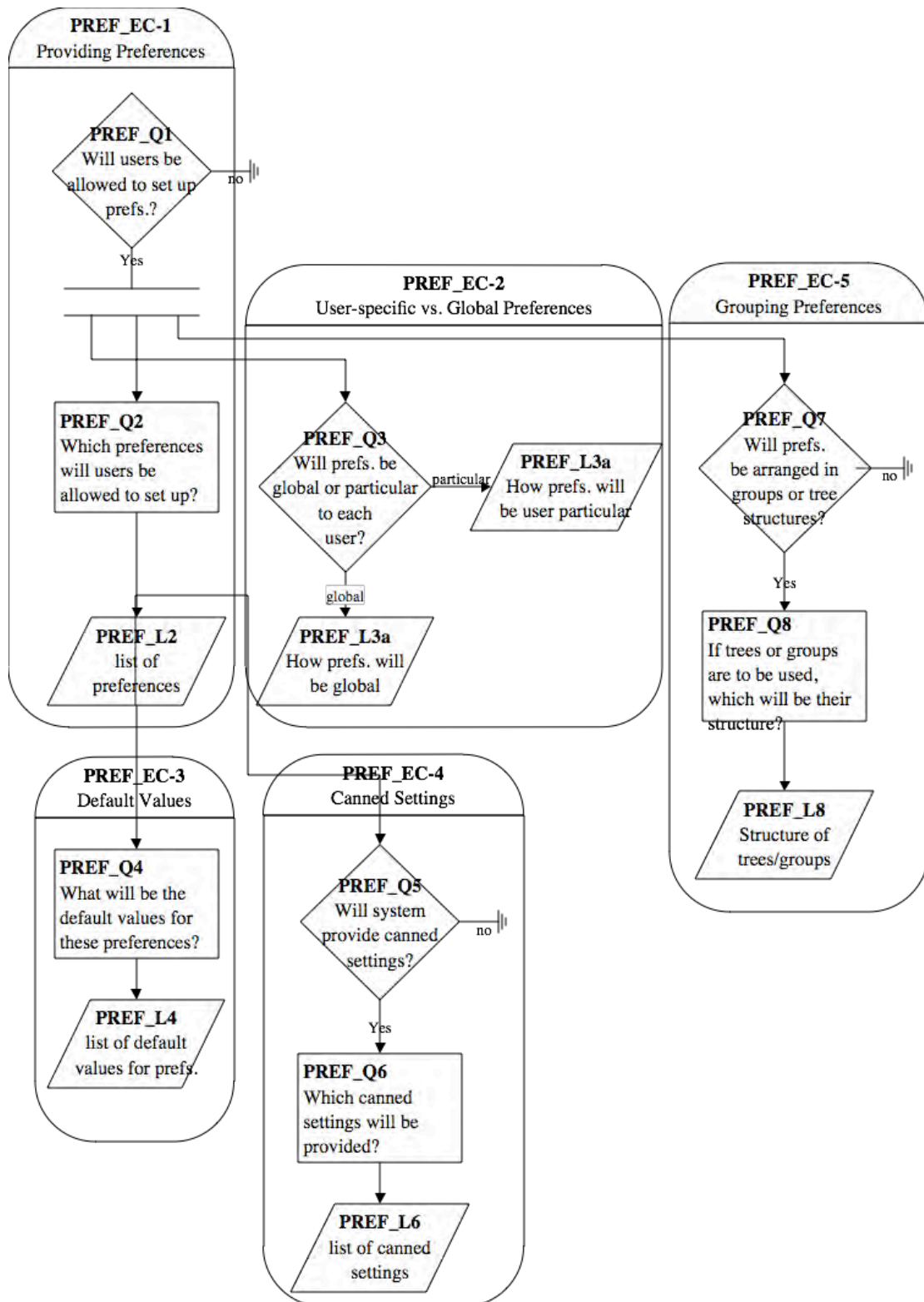


Figure 5.10-1: System Responsibility Clusters. Preferences.



### 5.10.1.3 Use Case Meta-model

The Use Case Meta-model for the Preferences feature is shown in Figure 5.10-2 (See page 71 for color legend), in which five use cases are identified and described below.

**PREF\_UC-1 SavePreferencesForUser:** Upon making changes to one or more preferences, the user requests for them to be saved. This will trigger the included use case PREF\_UC-4 StorePreferenceValuesToPersistence

**PREF\_UC-2 LoadCannedSettings:** The user requests a group of canned settings to be loaded, allowing him to set a number of preferences all at once. This use case triggers PREF\_UC-5 LoadPreferenceValuesFromPersistence when loading the canned setting to use.

**PREF\_UC-3 LoadPreferencesForUser:** The user requests his preferences to be loaded, directly or indirectly. For example, starting the application is an indirect way to request preferences to be loaded in certain systems. This use case also triggers PREF\_UC-5 LoadPreferenceValuesFromPersistence, as each of the preferences to load need to be ‘filled in’ with their value, stored in persistence.

**PREF\_UC-4 StorePreferenceValuesToPersistence:** This use case is triggered by PREF\_UC-1 SavePreferencesForUser every time a user choose to save his current preferences. It writes the preference values onto the predetermined physical medium.

**PREF\_UC-5 LoadPreferenceValuesFromPersistence:** This use case is triggered by PREF\_UC-2 LoadCannedSettings when loading a canned setting. Each of the default values for the preferences contained in that setting need to be loaded from persistence.

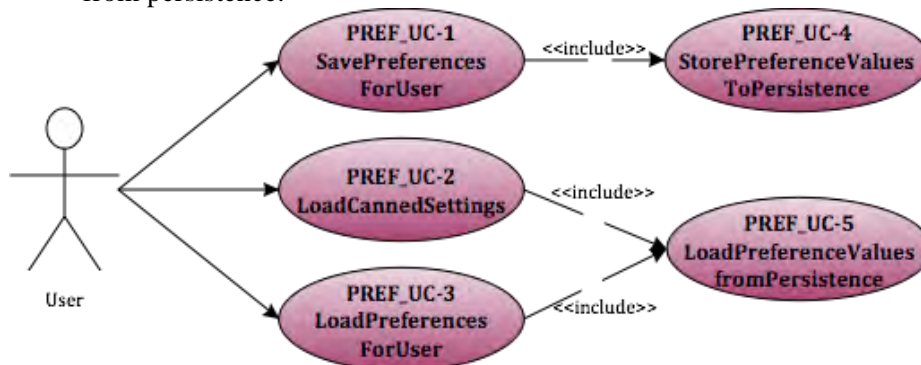


Figure 5.10-2 Use Case Model. Preferences

When using only global preferences (meaning all users see the same preferences as there are no per-user distinctions) the use cases remain the same, only instead of loading, storing and saving preferences for a particular user it is done so for the entire application (which can also be seen as a ‘master’ user if needed)

The applicability of each of these use cases will depend on the results of the elicitation process. If during elicitation of the Preferences Functional Usability Feature it were determined that ‘canned settings’ are not needed, for example, the LoadCannedSettings use case would be discarded. Use cases also depend on one another. These dependencies are shown in Table 5.10-2 where we can see the following:

- For the **Save Preferences For User** use case to be viable, it only needs its included use case Store Preference Values to Persistence, which writes the saved preferences into a physical medium.

- The **Load Canned Settings** use case needs the **Load Preference Values from Persistence** use case, which actually retrieves these settings from the physical medium in which they're stored.
- The **Load Preferences For User** use case also needs the **Load Preference Values from Persistence** use case for the same reason. It also requires that the **Save Preferences For User** use case exists, as if no preferences are ever saved there will be no preferences to load.
- The **Store Preference Values to Persistence** needs the **Save Preferences For User** use case for a similar reason, as nothing can be stored to persistence if it is not first ordered to be saved through the user interface.
- And finally, **Load Preference Values from Persistence** requires that the **Store Preference Values to Persistence** exist, because in order to be loaded, preference values must first be stored. It also needs, minimally, that the **Load Preferences For User** use case exist, as preferences won't be loaded unless they are ordered to through the user interface.

Table 5.10-2 Usability Use Case Dependencies: Preferences Functional Usability Feature

	PREF_UC-1 Save Prefs For User	PREF_UC-2 Load Canned Settings	PREF_UC-3 Load Prefs. For User	RREF_UC-4 Store Pref. Vals to Pers.	PREF_UC-5 Load Pref. Vals from Pers
PREF_UC-1 SavePreferencesForUser	-			X	
PREF_UC-2 Load CannedSettings		-			X
PREF_UC-3 Load Preferences For User	X		-		X
PREF_UC-4 Store Preference Vals to Persist.	X			-	
PREF_UC-5 Load Pref Vals from Persist.			X	X	-

By looking at the table columns, we can see that in spite of the fact that Save Preferences For User and Load Preference Values from Persistence have the higher number of dependencies from other use cases (two each). Thus, initially, no single use case is evidently core to the Preferences Functional Usability Feature at this stage. This fact makes the Preferences feature somewhat versatile, as multiple sub-groups of use cases could be correctly implemented.

#### 5.10.1.4 System Responsibilities for Usability

Table 5.10-3 shows the proposed System Responsibilities for Usability for the present feature.

Table 5.10-3 System Responsibilities List for Preferences

System Responsibilities List for Undo
PREF_SR-1 Set Preferences Saves preferences for a single user or for the application as a whole
PREF_SR-2 Provide default values Allows preferences to have pre-determined values in case none is chosen by the user
PREF_SR-3 Allow 'canned settings' Provides groups of preferences with pre-determined values to be loaded together
PREF_SR-4 Organize preferences Determines how preferences will be presented to the user, whether in groups or trees

These System Responsibilities are derived from the Usability Elicitation Clusters as follows:

**PREF\_EC-1 Providing Preferences:** This Elicitation Cluster contemplates whether preferences will be allowed to be set and saved by users and, if so, which ones they will be. Thus, this elicitation cluster yields System Responsibility **PREF\_SR-1 Set Preferences**, together with PREF\_EC-2, as explained as follows.

**PREF\_EC-2 User-specific vs. Global Preferences:** This Elicitation Cluster covers the need for allowing specific users to save their own preferences independently from other users, or to, otherwise, have a single set of preferences, shared by all

users. As such, this cluster also contributes to System Responsibility **PREF\_SR-1 Set Preferences**.

**PREF\_EC-3 Default Values:** This Elicitation Cluster covers determining the default values for each preference, directly yielding the System Responsibility **PREF\_SR-2 Provide Default Values**.

**PREF\_EC-4 Canned Settings:** From this Elicitation Cluster addresses the need for predefined sets of multiple preferences, and it translates into the sole System Responsibility **PREF\_SR-3 Allow ‘canned settings’** is derived.

**PREF\_EC-5 Grouping Preferences:** Finally, this elicitation cluster contemplates arranging preferences in groups or trees, and gives way to the **PREF\_SR-4 Organize Preferences** System Responsibility.

Table 5.10-4 maps these relationships between Usability Elicitation Clusters and the Usability System Responsibilities, for easy reference. Any project determined to require a specific Elicitation Cluster will also require its related System Responsibilities. Likewise, if a cluster is discarded during elicitation, its related responsibilities will not be a part of the resulting system.

Table 5.10-4 Usability Elicitation Clusters / System Responsibilities Mapping for Preferences

Use Cases	Dependent Responsibilities
PREF_EC_1 Providing Preferences	PREF_SR-1 Set Preferences
PREF_EC_2 User-specific vs. Global Preferences	PREF_SR-1 Set Preferences
PREF_EC_3 Default Values	PREF_SR-2 Provide Default Values
PREF_EC_4 Canned Settings	PREF_SR-3 Allow Canned Settings
PREF_EC_5 Grouping Preferences	PREF_SR-4 Organize Preferences

## 5.10.2 Usability Guideline for Software Development: Design artifacts

The design artifacts of the Usability Guideline for Software Development for the Warning feature are described in the following sections. The System Responsibilities described above are brought to a lower abstraction level as High-level Design Component Responsibilities in section 5.10.2.1. Section 5.10.2.2 expresses them as Low-level Design Component Responsibilities (for a MVC architecture). Finally, section 5.10.2.3 presents the Usability Design Meta-models for said Low-level Design Component Responsibilities as object-oriented class and sequence diagrams.

### 5.10.2.1 High-level Design Component Responsibilities

To support the System Responsibilities at design level, the following sections describe the suggested High-level Design Components for the Preferences feature, summarized in Table 5.10-5.

#### 5.10.2.1.1 Preference Component

This component is responsible for holding the basic data related to a ‘live’ preference. For example, in a system where a certain color attribute is set to ‘red’, there will be one instance of the Preference class holding the name of that color attribute, the possible values that it could take, and fact that it is currently taking the ‘red’ value.

#### 5.10.2.1.2 Settings Component

The Setting component represents a group of Preferences with an assigned value. It is a set of pairs, comprised of a preference name and a given value, intended to be saved and loaded as a group (when, for example, using canned settings)

### 5.10.2.1.3 PreferenceManager Component

This component is in charge of managing individual Preferences within the system. All operations to be performed on Preference objects must go through their PreferencesManager, as it is the class with the responsibility of knowing and handling their information.

### 5.10.2.1.4 StorageFront Component

The StorageFront Component is in charge of storing and retrieving preference values into persistence. It is the only component allowed such access to the physical media where these values are stored, and any classes needed to make use of them must present a request to do so to the StorageFront.

### 5.10.2.1.5 SettingsManager Component

This component is in charge of saving and loading Settings upon request within the system.

### 5.10.2.1.6 User Component

The User Component is in charge of holding and accessing a sole Settings component. This particular Settings component holds all the preference values stored for this particular User. The User class is the sole responsible for accessing these values, whether for loading or for saving.

### 5.10.2.1.7 Group Component

The Group component may hold one or many Preference objects in the system when simple grouping was elicited. To represent a tree structure, the Group object will hold either Preferences or Groups, as leaves and branches of said tree, respectively.

Table 5.10-5: Usability Guideline: High-level Design Component Responsibilities. Preferences

System Responsibility	Generic Component Responsibilities
PREF_SR-1 Set Preferences	<p>A <i>Preference</i> component holds the information related to a single 'live' system preference, minimally: its name (i.e. Background color), its possible values (i.e. green, red, blue) and the current active value it may have. A preference is 'live' once it's been loaded (as opposed to a preference setting that may be stored in the hard drive for later use)</p> <p>A <i>PreferencesManager</i> component is responsible for knowing, handling and retrieving all live <i>Preference</i> components within the system.</p> <p>A <i>Setting</i> Component represents a group of predetermined value pairs (preference name – preference value) that can be loaded from the hard drive (through the <i>Storage Front</i> Component) and rolled out into the live preferences.</p> <p>The <i>StorageFront</i> component represents the link between the application logic and the repository where the preference values are saved. A <i>Setting</i> will load its values through the <i>StorageFront</i>, as only this class has direct access to the information stored in the hard drive (or other media).</p> <p>A <i>User</i> component represents a system user if different users are required to hold different preferences (as opposed to having global preferences for the entire application). The <i>User</i> is responsible also for holding it's own default <i>Setting</i>, if one exists.</p>
PREF_SR-2 Provide default values	The <i>Preference</i> component is also responsible for knowing what (if any) is its default value and for setting itself to that value if/when requested by the <i>UI</i>
PREF_SR-3 Allow 'canned settings'	A <i>SettingsManager</i> is responsible for loading stored <i>Settings</i> when asked by the <i>UI</i>
PREF_SR-4 Organize preferences	If preferences are to be grouped, a <i>Group Component</i> is responsible for holding related preferences and for providing the <i>UI</i> with them.

### 5.10.2.2 Low-level Design Component Responsibilities for MVC

When instantiating for a MVC architecture, the **UI** Component is instantiated by View the object(s) and takes over all of its responsibilities except for the delegation of actions to other objects. This responsibility falls on the **Controller** object(s) of the MVC architecture.

The Preference Component is represented by **Preference** class and covers all of its responsibilities. It holds the currently assigned (or active) value that the user has last set it to, or that has been loaded from persistence. Preference objects are always contained within a Setting object, described below.

The Setting Component is represented by the **Setting** class, and it is responsible for managing Preference objects. So called ‘canned settings’ are represented, for example, by a single Setting object containing a certain number of preferences with an assigned value. Such a setting would be loaded or ‘rolled out’ when a user calls for a ‘canned setting’ to be loaded. Another use for the Setting object is to contain a single User’s (or the application’s) desired preference values. In multi-user systems, each User will contain a single Setting object, holding and managing all of its preferences at all times.

The PreferencesManager Component is represented by the **PreferencesManager** class, and covers all its responsibilities. It is responsible for ordering modified Preferences to be saved, and to retrieve them when requested by the View.

The StorageFront Component is represented by the **StorageFront** parent class, and by any of its subclasses. These classes are responsible for storing any saved data to a given physical medium. Each subclass of **StorageFront** implements this functionality for each needed particular physical medium. For example, if only a database is to be used, the StorageFront component would have a DatabaseStorageFront subclass implementing this functionality. If later on it becomes necessary to provide additional storing capabilities unto text files, for example, a TextFileStorageFront subclass would be created and used as needed for this purpose.

The SettingsManager Component is represented by the **SettingsManager** class, and covers all of the components responsibilities. It is responsible for ordering newly created settings to be saved, and to retrieve them when requested by the View.

The User Component’s responsibilities are carried out by the **User** class, in charge of holding and managing its Setting object, containing all of the user’s preferences.

Finally, the Group Component is represented by the **Group** class for organizing and arranging Preferences in the desired structure for display in the View.

Table 5.10-6 details the Low-level Design Component Responsibilities described above and how they carry out each of the System Responsibilities for Usability. For each of these, the sequence of actions required by the different objects is presented as well as a set of UML diagrams that depict each of these interactions.

Table 5.10-6: Usability Guideline: Low-level Design Component Responsibilities (MVC). Preferences.

System Responsibility	Objects							Fig
	View	Controller	User	Setting	StorageFront	Preference	Preferences Manager	
PREF_SR-1 Set Preferences	1. The <i>View</i> listens for user calls to save a group of changed preferences. If they belong to individual user, <i>userId</i> is sent. It forwards the request to the <i>Controller</i>	2. The <i>Controller</i> forwards the request and list of preference values to the appropriate <i>User</i> (or a <i>Preferences Manager</i> if 'global' preferences)	3. The <i>User</i> requests its <i>Setting</i> object to update all of the sent preference values	4. For each value sent in <i>prefVals[]</i> , the <i>Setting</i> object within the <i>User</i> updates its internal <i>Preference</i> objects. 6. The <i>Setting</i> saves itself once all of its internal <i>Preferences</i> have been updated	5. The <i>StorageFront</i> stores the saved <i>Setting</i> object to the appropriate persistence medium.	5. Each <i>Preference</i> object updates its value to be the new value sent.		Figure 5.10-5
	1. The <i>View</i> listens for user calls to load his prefs (or the system's, in case of 'global')	2. The <i>Controller</i> requests the appropriate <i>User</i> (or a <i>Preferences Manager</i> if 'global' ) the <i>Setting</i> object and requests it to <i>rollOut()</i>		3. The <i>Setting</i> object rolls itself out by loading all of its pref-value pairs from the <i>StorageFront</i> and loading them onto the 'live' <i>Preferences</i> , via the <i>PreferencesManager</i> . 5. For each pref name in the received pref-value pairs, the <i>Setting</i> asks the <i>PreferencesManager</i> to update the corresponding <i>Preference</i> object (live) with the designated value.	4. The <i>StorageFront</i> loads all of the pref-value pairs belonging to the <i>Setting</i> in question and returns them to it	7. The <i>Preference</i> object updates itself as requested by the <i>Preferences Manager</i>	6. The <i>Preferences Manager</i> orders each <i>Preference</i> to update itself with the new value	Figure 5.10-4
PREF_SR-2 Provide default values	1. The <i>View</i> listens for user calls to reset the default values for a group of preferences. It forwards the call to the <i>Controller</i>	2. The <i>Controller</i> requests the appropriate user (or <i>PreferencesManager</i> in case of 'global') to reset the group of prefs. to its defaults	3. The <i>User</i> forwards the call to its <i>Setting</i> object	4. The <i>Setting</i> orders each <i>Preference</i> to reset itself to its <i>defaultValue</i> 6. The <i>Setting</i> saves itself via the <i>StorageFront</i>	7. The <i>Storage Front</i> writes the updated <i>Setting</i> to the appropriate storage medium.	5. The <i>Preference</i> object, responsible for knowing and setting its default value, sets its <i>currentValue</i> to that default		Figure 5.10-7
PREF_SR-3 Allow 'canned settings'	1. The <i>View</i> listens for user calls to load a canned setting. It forwards this request, along with the <i>settingId</i> to the <i>Controller</i>	2. The <i>Controller</i> requests the <i>Settings Manager</i> the <i>Setting</i> object and requests it to <i>rollOut()</i>		3. The <i>Setting</i> object rolls itself out by loading all of its pref-value pairs from the <i>StorageFront</i> and loading them onto the 'live' <i>Preferences</i> , via the <i>PreferencesManager</i> . 5. For each pref name in the received pref-value pairs, the <i>Setting</i> asks the <i>PreferencesManager</i> to update the corresponding <i>Preference</i> object (live) with the designated value.	4. The <i>StorageFront</i> loads all of the pref-value pairs belonging to the <i>Setting</i> in question and returns them to it	7. The <i>Preference</i> object updates itself as requested by the <i>Preferences Manager</i>	6. The <i>Preferences Manager</i> orders each <i>Preference</i> to update itself with the new value	Figure 5.10-6
PREF_SR-4 Organize preferences	1. When preferences are loaded, the <i>View</i> is responsible for displaying them as groups/trees if applicable							Figure 5.10-4

### 5.10.2.3 Usability Software Design Meta-models

These UML diagrams represent the Low-level Design Component Responsibilities described in earlier. The following sections describe the class diagram and the classes involved in this feature and their interrelationships, followed by the descriptions of the sequence diagrams.

#### 5.10.2.3.1 Class Diagram

Figure 5.10-3 below shows the class diagram for the Preferences feature. The main objects involved are the View, Controller, Preference, Group, PreferencesManager, User, Setting, SettingsManager, StorageFront, StaticFileStorageFront and DatabaseStorageFront.

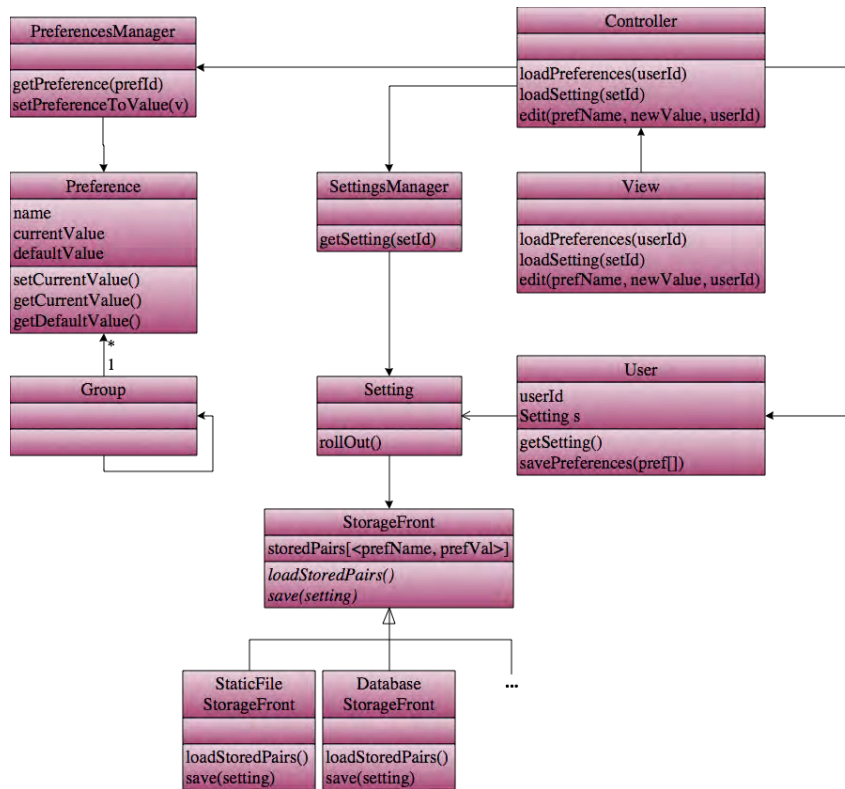


Figure 5.10-3: Usability Design Meta-model. Class diagram. Preferences.

#### 5.10.2.3.2 Sequence Diagram "Load Preferences"

Figure 5.10-4 shows the sequence diagram for loading preferences, covering the Low-level Design Component Responsibilities of PREF\_SR-1. This sequence starts when the user (directly or indirectly) requests to load his preferences. This request is forwarded to the controller who knows that a User's preferences are stored in its Setting object. As such, the Controller locates the appropriate User, and invokes its Setting object to rollOut(), meaning it will load all the necessary preference values for the Controller to then do what it must with them. For example, if 'Language = Spanish' is one of the preference-value pairs that are loaded, the Controller will continue execution by performing the actions needed to display Spanish text. This is obviously outside of the scope of the present feature as it depends solely on the domain-specific requirements for the project.

In order for the Setting to be rolled out, it must order the StorageFront to load its stored preference-value pairs. Once these are loaded, it orders the PreferencesManager to load these values onto the active Preferences one by one. Once the active preferences are loaded with the stored values, the control returns to the Controller object, who moves on to taking any further actions, if needed.

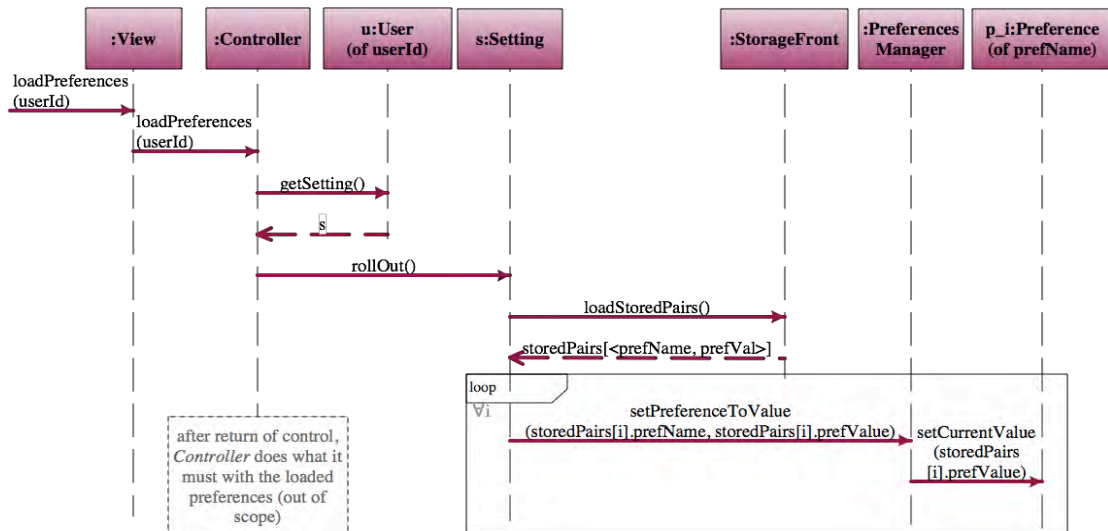


Figure 5.10-4: Sequence Diagram "Load Preferences". Preferences.

#### 5.10.2.3.3 Sequence Diagram "Save Preferences"

Figure 5.10-5 shows the sequence diagram for saving preferences. This diagram covers all the object responsibilities listed for PREF\_SR-1.

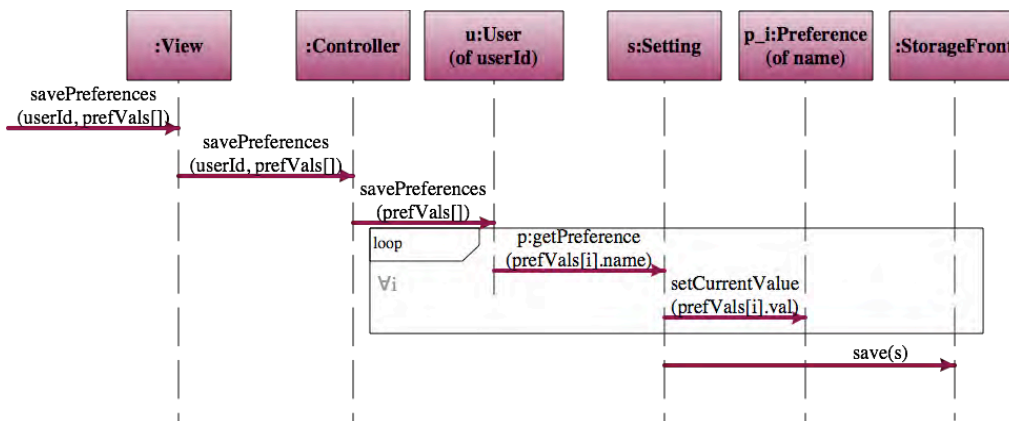


Figure 5.10-5: Sequence Diagram "Save Preferences". Preferences.

This sequence starts when the user requests for his preferences to be saved (for example, after having changed them). The View forwards this request to the Controller, with the `userId` (if applicable) and an array of preference-value pairs. As it is the User object's responsibility to store its own preference values (inside its Setting object), the Controller forwards this information to the User of `userId`, invoking its `savePreferences` method.

For every preference-value pair passed to the User, the corresponding Preference object is retrieved from within the User's Setting object. That Preference's value attribute is then replaced by whatever value was contained in the aforementioned preference-value pair. Once all Preferences in the User's Setting have been updated with the new values, the newly altered Setting is ordered to be saved to persistence by the appropriate StorageFront.

#### 5.10.2.3.4 Sequence Diagram "Load Canned Setting"

This sequence covers the System Responsibility PREF\_SR-1 and is shown in Figure 5.10-6. It is nearly identical to the one described in section 5.10.2.3.2 Sequence Diagram "Load Preferences", except the preferences being loaded in this case are not those saved for a particular user, but rather a 'canned setting'. This difference can be seen graphically in the call to `getSetting(settingId)` not being made to a User object, but rather to the SettingsManager, in charge of loading these type of system-wide settings. This called is



triggered by the user requesting to load a particular canned setting and results in that setting object (canned\_s) being returned to the Controller

Once the Controller has the Setting object it orders it to roll out, meaning that said setting will order the StorageFront to load all of its preference-value pairs from persistence. For each of these pairs being loaded, the Setting will ‘fill in’ the values in the ‘live’ Preferences, effectively loading them as requested. Once the active preferences are loaded with the values for these canned settings, the control returns to the Controller object, which moves on to taking any further actions, if needed.

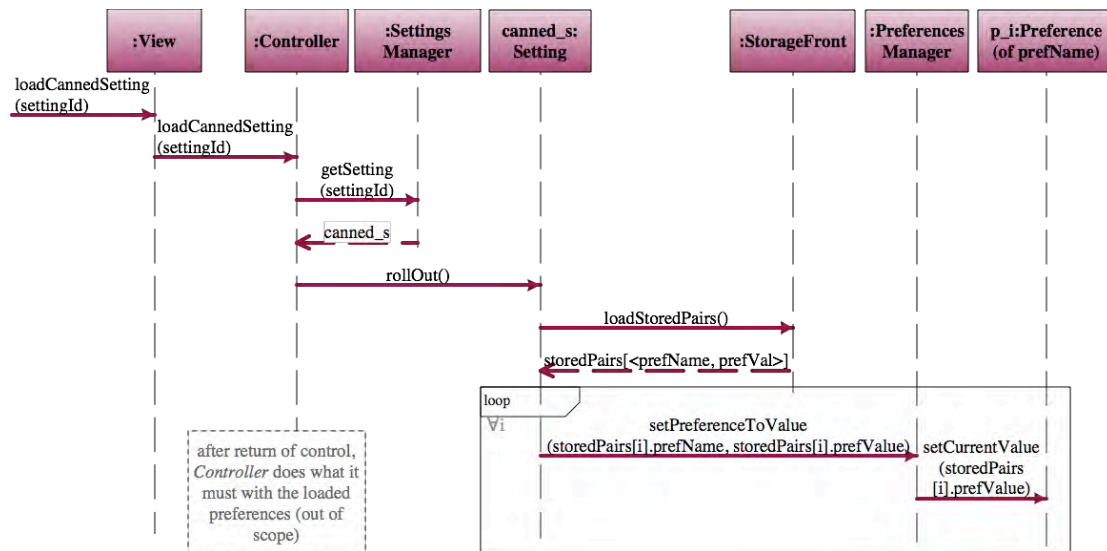


Figure 5.10-6: Sequence Diagram "Load Canned Settings". Preferences

### 5.10.2.3.5 Sequence Diagram "Reset Default Values"

This sequence, shown in Figure 5.10-7, starts when the user requests the View to reset the default values to a set of preferences. This can be a single preference being reset to its default value, a few selected preferences needing to be reset, or resetting all preferences to default.

The request to reset the selected group of preferences is forwarded to the controller who, in turn, requests the appropriate user to reset those preferences for itself. Similarly to saving regular preferences for a user, resetting the default value for preferences means that for every preference passed to the User, the corresponding Preference object is retrieved from the User's Setting object. That Preference's current value attribute is then replaced by its default. Once all the selected Preferences have been reset, the newly altered Setting is ordered to be saved to persistence by the appropriate StorageFront.

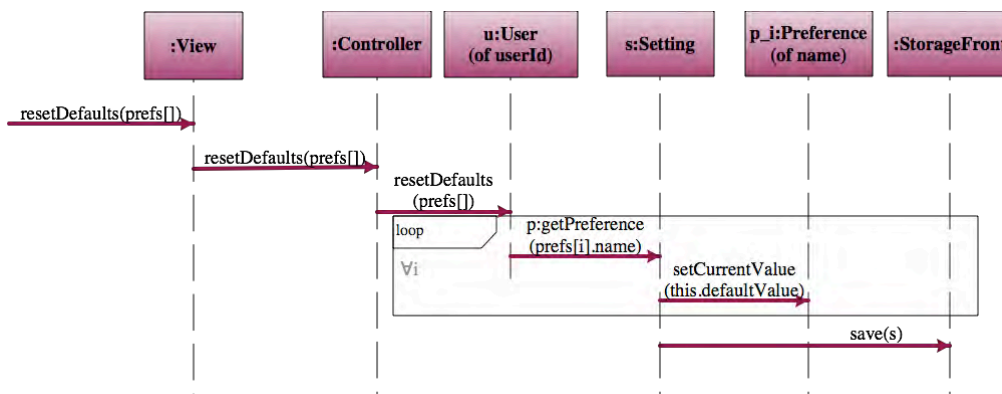


Figure 5.10-7: Sequence Diagram "Reset Default Values". Preferences

## 5.11 “Favorites” Usability Guideline for Software Development

The Favorites Functional Usability Feature covers the user’s need to bookmark and keep a collection of favorite places within an application. When the array of places that users can visit within an application is vast, they will need a way to remember those of interest to them and retrieve them easily at a later time.

The Usability Guideline for Software Development is made up of Analysis and Design artifacts, described for the Warning feature in the following two sections.

### 5.11.1 Usability Guideline for Software Development: Analysis artifacts

There are four artifacts to be used during the analysis phase: the Usability Elicitation Guideline, the Usability Elicitation Clusters, the Usability Use Case Meta-models, the System Responsibilities for Usability. These artifacts are described in the following four sections.

#### 5.11.1.1 Usability Elicitation Guideline

Table 5.11-1 shows the Usability Elicitation Guideline for the Favorites Functional Usability Feature. There are two HCI recommendations in this guideline, covering what the Favorites list is and how it should be structured. These recommendations are explained below.

##### 5.11.1.1.1 Favorites List

HCI authors suggest that users need to record their points of interest within an application in order to return to them later (F\_HCI-1). Points of interest can lie within the application itself (i.e. marking an email as ‘important’ within an email program, to easily identify it and return to it later) or beyond the scope of the application (i.e. a web bookmark, within the application, that loads an external website). This subtle but significant difference should be carefully noted in each case during the elicitation phase (F\_ELAB-1). Discussions should revolve around defining a bookmark (or ‘favorite’), how they will be stored, what information they will hold, how they will be retrieved and the maximum number (if any) of bookmarks that can be saved in the application (F\_Q-1 to F\_Q-5).

Table 5.11-1, example F\_EX-1 “Favorites in Navigation” describes an example for this HCI recommendation.

##### 5.11.1.1.2 Favorites List Structure

HCI authors suggest that if the list of bookmarks gets too long, the user should be allowed to structure it or rank it for ease of access (F\_HCI-2). The most common way to organize a list of bookmarks is by allowing the user to sort them in folders. Another, more recent sorting technique is the tagging of bookmarks (F\_ELAB-2). Stakeholders must discuss which means the system must provide the user for sorting bookmarks (F\_Q-4 to U\_Q-6).

Example F\_EX-2 “Browser Bookmarks” in Table 5.11-1 describes an example for this HCI recommendation.

Table 5.11-1. Usability Requirements Elicitation Guideline. Favorites.

Identification			
Name	Favorites		
Family	User Profile		
Aliases	Bookmarks [42]; Favourites [49]		
Intent			
Providing a mechanism for bookmarking and keeping a collection of such favorite places within an application			
Problem			
When the array of places that users can visit within an application is vast, they will need a way to remember those of interest to them and retrieve them easily at a later time.			
Context			
In a navigable software system, when the system is possibly large and complex and allows the user to move freely through it in ways not directly supported by the artefact's structure.			
HCI Recommendation	Elaboration	Discussions with Stakeholders	Usage Examples (optional)
<p>F_HCI-1 Favorites list</p> <p>Let the user make a record of their points of interest, so that they can easily go back to them later. The user should be able to label them, since users are in a better position to choose labels that are memorable to them. Save the list for later use [42].</p> <p>Users need to temporarily gather a set of items for later use. Allow users to build their list of items by selecting the items as they are viewing them. [49]</p>	<p>F_ELAB-1 Favorites scope</p> <p>Points of interest, or bookmarks, can be more than simple locations within the application. They can also be outside of the scope of the application as is the case of a Favorite's list of URLs gathered within a "home" website.</p>	<p>F_Q-1 Will user be allowed to collect bookmarks of locations into a "Favorites" list?</p> <p>F_Q-2 If so, what constitutes a bookmark?</p> <p>F_Q-3 How will the user add these bookmarks to the Favorites list?</p> <p>F_Q-4 How will the user retrieve the contents of the bookmarked item?</p> <p>F_Q-5 How many bookmarks is the user allowed to add to the Favorites list?</p>	<p>F_EX-1 Favorites in navigation</p> <p>In applications where users navigates through (ordered or unordered) "spaces", not having a means to keep track of places that s/he would like to revisit can be frustrating for the user. A lack of such a mechanism would limit the user's potential within the application (if e-commerce site, for example, user may not remember items s/he intends to buy later)</p>
<p>F_HCI-2 Structure of the Favorites list</p> <p>If the list becomes long, allow users to structure it [49]. Support at least an ordered linear organization, so that a user can rank them according to whatever criteria they choose; if possible, support a grouping structure of some kind [42].</p>	<p>F_ELAB-2 Structure: Folders and Tags</p> <p>The grouping of bookmarks is usually done by means of either "folders" or "tags". In the case of folders, a bookmark <i>belongs</i> to the folder where they are placed, and can only exist in one folder at a time.</p> <p>Tags, conversely, are assigned bookmarks. One bookmark can have many tags, and the same tag can be assigned to many bookmarks.</p>	<p>F_Q-6 Will the Favorites list be structured?</p> <p>F_Q-7 If so, will structuring be done linearly or in groups?</p> <p>F_Q-8 If linearly, what will be the ordering criteria?</p> <p>F_Q-9 If grouped, will grouping be static (folders) and/or dynamic (tags)?</p> <p>F_Q-10 How will a bookmark be added to a folder (if applicable)?</p> <p>F_Q-11 How will a bookmark be tagged (if applicable)?</p>	<p>F_EX-2 Browser Bookmarks</p> <p>Once a favorites list contains more than just a few browser bookmarks, finding a bookmark becomes a daunting task. The usefulness of the list is diminished by the user's inability to find something they've previously saved.</p>

### 5.11.1.2 Usability Elicitation Clusters

The Usability Elicitation Guideline suggests nine discussion items to be held with stakeholders in order to elicit all aspects of the Favorites Functional Usability Feature. These discussion items can be clearly divided into six clusters, as described in the Usability Elicitation Cluster Map in shown in Figure 5.11-1, according to the portion of the Favorites functionality that they cover.

**F\_EC\_1 Adding bookmark to favorites list:** The discussion items in this cluster cover determining whether favorites will be needed in the system and if so, what they will be and how they'll be collected.

**F\_EC\_2 Retrieving bookmark:** Once it is established what constitutes a bookmark, stakeholders must define how its contents will be retrieved.

**F\_EC\_3 Structuring favorites list:** These discussions deal with determining, for large sets of bookmarks, how they will be organized. Options include linear lists and groups, where groups can be dynamic or static (see below)

**F\_EC\_4 Determining order criteria:** If lists are to be linear, it must be determined in which ways they can be sorted. These discussions deal with the different criteria that might be used for doing so.

**F\_EC\_5 Grouping bookmarks:** If stakeholders decide that bookmarks will be grouped statically (into classic folders), it must be determined how a bookmark or group of bookmarks will be added to such groups.

**F\_EC\_6 Tagging Bookmarks:** If tags are to be used for grouping bookmarks (dynamically), stakeholders must determine how tags will be associated with bookmarks, i.e. how tagging will take place.

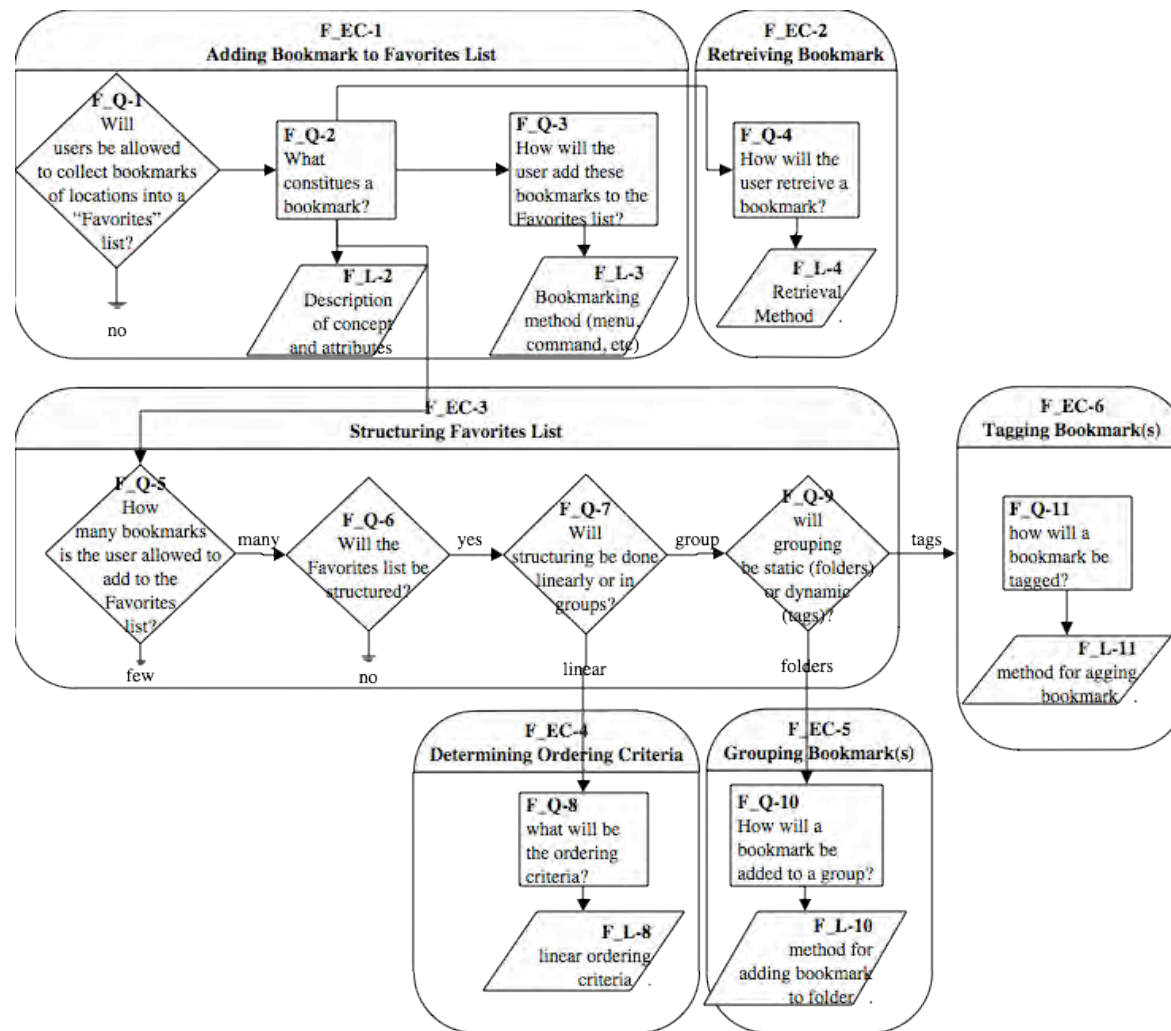


Figure 5.11-1: Elicitation Clusters. Favorites

### 5.11.1.3 Use Case Meta-model

The Use Case Meta-model for the Favorites feature is shown in Figure 5.11-2 (See page 71 for color legend), in which six use cases are identified and described below.

**F\_UC\_1 CreateFavorite:** This use case begins when the user requests to bookmark an object. This implies adding it to the favorites list (AddToList) and can entail adding it to a group (Group) or tagging it (Tag)

**F\_UC\_2 AddToList:** The CreateFavorite use case triggers this use case. It entails adding a newly created bookmark to the favorites list.

**F\_UC\_3 Tag:** A tag or tags are associated to the newly created bookmark. Many tags can be associated with one or more bookmarks.

**F\_UC\_4 Group:** The newly created bookmark is added to an existing group. A bookmark can only be added to a single group at a time

**F\_UC\_5 ViewFavorites:** The user requests to see a full list of all the saved favorites. Upon displaying the list, ViewFavorites must first call for the bookmarks to be sorted first, as determined at elicitation time (if applicable)

**F\_UC\_6 SortFavorites:** This use case is invoked from within ViewFavorites and involves sorting the bookmarks as requested for display.

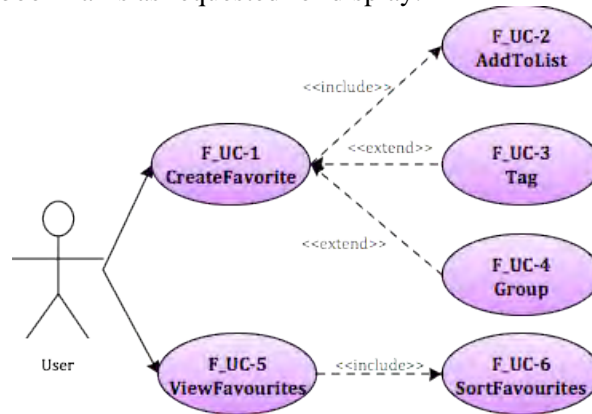


Figure 5.11-2 Use Case Meta-model. Favorites

As mentioned above, the applicability of each of these use cases will depend on the results of the elicitation process. If during elicitation of the Favorites Functional Usability Feature it is determined that, for example, no tags will ever be used, then the F\_UC-3 use case will be excluded from the model. Use cases also depend on one another. These dependencies are shown in Table 5.11-2 we can see the following

- The **CreateFavorite** use case will invariably need the **AddToList** use case, as the newly created bookmark will need to be stored after creation. Creating a favorite may require tagging or grouping, but neither action would impede this use case.
- **ViewFavorites** will need the **SortFavorites** use case to be viable (conditionally, hence the asterisk) only in the instances where some kind of sorting was determined to be required at elicitation time.
- Finally, every use case will need **CreateFavorite** because a bookmark must exist in order for these three use cases to be viable

Table 5.11-2 Usability Use Case Dependencies: Favorites Functional Usability Feature

	F_UC-1 Create Favorite	F_UC-2 Add ToList	F_UC-3 Tag	F_UC-4 Group	F_UC-6 View Favorites	F_UC-6 Sort Favorites
F_UC-1 CreateFavorite	-	X				
F_UC-2 AddToList	X	-				
F_UC-3 Tag	X		-			
F_UC-4 Group	X			-		
F_UC-6 ViewFavorites	X				-	X*
F_UC-6 SortFavorites	X					-

By looking at the columns, it becomes evident that the CreateFavorite use case is core to the Favorites feature. Furthermore, all but the CreateFavorite and AddToList use cases could potentially be discarded and still the Favorites feature would be viable (bookmarks would only be created and stored). This would be the minimal expression of this feature.

#### 5.11.1.4 System Responsibilities

Table 5.11-3 shows the proposed System Responsibilities for the Warning feature.

Table 5.11-3 System Responsibilities List for Favorites

System Responsibilities List for Undo	
F_SR-1	Create Bookmark The system must store all bookmarks upon creation by the user
F_SR-2	Retrieve Bookmark The system provide a way for retrieving each bookmark's content
F_SR-3	Set order criteria The system must provide ways for the user to sort the bookmarks
F_SR-4	Group bookmark(s) The system must allow users group bookmarks in folders
F_SR-5	Tag bookmark(s) The system must provide a way to tag bookmarks

These System Responsibilities are derived from the Usability Elicitation Clusters as follows:

**F\_EC\_1 Adding bookmark to favorites list:** This cluster contemplates creating a bookmark and saving it. It yields one System Responsibility: **F\_SR-1 Create Bookmark.**

**F\_EC\_2 Retrieving bookmark:** This cluster holds a single stakeholder discussion, regarding the method through which bookmarks will be retrieved, giving way to a single System Responsibility: **F\_SR-2 Retrieve Bookmark.**

**F\_EC\_3 Structuring favorites list:** This cluster contains discussion items aimed at defining the exact structure of favorites list. It does not yield a System Responsibility on its own, but rather in conjunction with the next three clusters, which deal with the specific types of structuring available.

**F\_EC\_4 Determining order criteria:** Together with F\_EC-3, this elicitation cluster yields the System Responsibility: **F\_SR-3 Set order criteria.** If it is determined that the favorites list will be structured linearly (as opposed to in groups), this cluster will determine what the ordering criteria will be

**F\_EC\_5 Grouping bookmark(s):** Together with F\_EC-3, this elicitation cluster yields the System Responsibility: **F\_SR-4 Group Bookmark(s).** If the favorites list is to be structured in groups, the discussion items in this cluster will determine the method for grouping bookmarks statically (in folders)

**F\_EC\_6 Tagging bookmark(s):** Together with F\_EC-3, this elicitation cluster yields the System Responsibility: **F\_SR-5 Tag Bookmark(s).** If the favorites list is to be structured in groups, this cluster will determine how bookmarks will be grouped dynamically, through assigned tags.

Table 5.11-4 maps these relationships between the Usability Elicitation Clusters and the Usability System Responsibilities for easy reference. Any project determined to require a specific cluster will also require its related System Responsibilities. Likewise, if a cluster is discarded during elicitation, its related responsibilities will not be a part of the resulting system.

Table 5.11-4 Use Case/ System Responsibilities Mapping for Favorites

Use Cases	Dependent Responsibilities
F_EC-1 Adding bookmark to favorites list	F_SR-1 Create Bookmark
F_EC-2 Retrieving bookmark	F_SR-2 Retrieve Bookmark
F_EC-3 Structuring favorites list	F_SR-3 Set order criteria F_SR-4 Group Bookmarks F_SR-5 Tag Bookmarks
F_EC-4 Determining order criteria	F_SR-3 Set order criteria
F_EC-5 Grouping bookmark(s)	F_SR-4 Group Bookmarks
F_EC-6 Tagging bookmark(s)	F_SR-5 Tag Bookmarks

## 5.11.2 Usability Guideline for Software Development: Design artifacts

The design artifacts of the Usability Guideline for Software Development for the Favorites feature are described in the following sections. The System Responsibilities described above are brought to a lower abstraction level as High-level Design Component Responsibilities in section 5.11.2.1. Section 5.11.2.2 expresses them as Low-level Design Component Responsibilities for MVC. Finally, section 5.11.2.3 shows the Usability Design Meta-models.

### 5.11.2.1 High-level Design Component Responsibilities

To support the System Responsibilities at design level, the following sections describe the suggested High-level Design Components for the Warning feature, shown in Table 5.11-5.

#### 5.11.2.1.1 User Interface (UI) Component

This component is responsible for capturing all user invocations and forwarding them (possibly through a delegating component) to the appropriate part of the domain, usually that responsible for executing the invoked action. Specifically for this feature, calls to create a new bookmark or to view the bookmarks list are captured by the UI Component. The UI Component is also responsible for relaying information to the user, including appropriate feedback after an action has been executed, as is displaying the bookmarks list when requested.

#### 5.11.2.1.2 Domain Component

A Domain Component represents the part of the system that is ultimately responsible for executing the actions requested by the user. In this case, the domain component represents the bookmarked object, if applicable.

For example, in the case of a browser with bookmarks that point to existing websites, all the information about the bookmark is contained within it, and the bookmarked element (the site) lies outside of the scope of the feature. In such a case, no DomainComponent would be needed.

In a contrasting example, when working within photo organizing software like Picasa or iPhoto, favorites tend to be pictures or groups of pictures, which are objects that do indeed lie within the scope of the feature and must be represented by the appropriate DomainComponent.

#### 5.11.2.1.3 Bookmark Component

The Bookmark component holds all the information pertaining to the object being bookmarked. It is responsible for controlling this information, including any tags that may be assigned to it

#### 5.11.2.1.4 Group Component

The Group Component may hold many Bookmarks, yet a single Bookmark may belong to only one Group at a time. Groups can belong to Groups in the same way, providing for nested grouping (trees).



### 5.11.2.1.5 Tag Component

The Tag Component holds a tag's information. One tag can be associated to many Bookmarks at any one time and vice-versa.

### 5.11.2.1.6 Favorites List Component

This component is responsible for holding all of the Bookmarks and/or Groups of Bookmarks when applicable. It is also responsible for setting the ordering criteria as requested by the user. No more than one instance of this component should exist within the system at any given time.

Table 5.11-5: Usability Guideline: High-level Design Component Responsibilities. Favorites

System Responsibility	Generic Component Responsibilities
F_SR-1 Create bookmark	<p>The UI Component is responsible for listening for user calls to create a new bookmark. It is also responsible for gathering the information about the newly created bookmark. This information must entail the minimum bookmark details (i.e. name, address) but can also contain any groups the user wishes to add the bookmark to or any tags the user wishes to append to the bookmark. Additional descriptive information is also encouraged.</p> <p>The delegating component (if any) must forward the call to create a bookmark along with the information to a FavoritesList Component. The FavoritesList Component is responsible for keeping bookmarks organized and for their storage/retrieval</p>
F_SR-2 Retrieve bookmark	<p>The UI Component is responsible for listening for user calls to load an existing bookmark. To do so, the user could provide specific information about a single bookmark (i.e. name) or information regarding tags or groups that may yield a list of more than one bookmark.</p> <p>In either case, a delegating component (if any) forwards the call and the data to the Favorites List Component for retrieval</p> <p>The Favorites List Component uses the data provided to filter through its bookmarks and returns only those that match the user's request</p> <p>The UI Component (through a delegating component, if any) receives the list of bookmarks and prompts the user to select the one s/he wishes to load</p> <p>Upon selection, the UI is also responsible for loading the chosen bookmark (i.e. if it's a URL, the UI is responsible for making the necessary calls to domain classes--outside the scope of this pattern--to load the desired website)</p>
F_SR-3 Structure Favorites List: Set Ordering Criteria	<p>The Favorites List Component is responsible for keeping its bookmarks in the order specified by the user. It's responsible for providing different means of organization for the user to choose from .</p> <p>The UI Component is responsible for listening for user calls to set a new ordering criterion</p> <p>The delegating component (if any) must forward the call to the Favorites List, which will set the new ordering criterion as 'active'</p>
F_SR-4 Structure Favorites List: Group Bookmark(s)	<p>Bookmarks can be added to groups at create-time or after-the-fact. In the latter case, the UI Component listens for user calls to add an existing bookmark to an existing group.</p> <p>The delegating component (if any) locates the appropriate group and orders it to add (unto itself) said bookmark</p> <p>Groups can hold many bookmarks and one bookmark can reside in only one group at a time.</p>
F_SR-5 Structure Favorites List: Tag Bookmarks	<p>Bookmarks can be tagged at create-time or after-the-fact. In the latter case, the UI Component listens for user calls to add a tag (or tags) to an existing bookmark.</p> <p>The delegating component (if any) locates the appropriate bookmark and orders it to add (unto itself) the tags introduced by the user.</p> <p>Tags are assigned to bookmarks. One bookmark can have many tags and one tag can be assigned to many bookmarks</p>

### 5.11.2.2 Low-level Design Component Responsibilities for MVC

When instantiating for a MVC architecture, the UI Component is instantiated by **View** the object(s) and takes over all of its responsibilities except for the delegation of actions to other objects. This responsibility falls on the **Controller** object(s) of the MVC architecture.

The DomainComponent is represented by **BookmarkedElement**, and represents the object that a bookmark may link to if it lies within the scope of the application, as explained earlier.

The **Bookmark**, **Tag** and **Group** components are represented by the objects of the same name, and cover all of their functionalities, respectively.

Table 5.11-6 details the Low-level Design Component Responsibilities described above and how they carry out each of the System Responsibilities defined in section 5.2.1.4. For each System Responsibility, the sequence of actions required by the different objects is presented as well as a set of UML diagrams that depict each of these interactions.

Table 5.11-6: Usability Guideline: Low-level design component responsibilities (MVC). Favorites.

System Responsibility	Objects					Fig
	View	Controller	Bookmark	FavoritesCollection	Group	
F_SR-1 Create bookmark	1. The View listens for calls from the user to create a <code>newBookmark()</code> . Aside from the <code>bookmarkInfo</code> , said call can include a group to add the bookmark to and a list of <code>tags[]</code> to add to the new bookmark 2. The View passes on the information and the call to the Controller	2. The Controller, in turn, orders the <i>FavoritesCollection</i> to create the new bookmark with this information	6. The <i>Bookmark</i> adds as many <code>tags[]</code> as are passed on unto itself	3. The <i>FavoritesCollection</i> creates a new <i>Bookmark</i> object with the corresponding <code>bookmarkInfo</code> . 4. If a group was sent along in the call, the newly created bookmark to said group. If no group was sent, then the bookmark is added to a default group. 5. If <code>tags[]</code> were sent along, the <i>FavoritesCollection</i> passes them on to the newly created <i>Bookmark</i> . If no tags were sent, no tags are added to the <i>Bookmark</i> .		Figure 5.11-4
F_SR-2 Retrieve bookmark	1. The View listens for calls from the user to <code>get()</code> existing bookmarks according to a given criteria and forwards them to the Controller. 5. The View prompts the user to select, from among the received <code>bookmarks[]</code> , the one that s/he wishes to load 6. When the user selects a bookmark, the View makes the necessary method calls to load the <i>Bookmark</i> .	2. The Controller, in turn, orders the <i>FavoritesCollection</i> to load all the bookmarks (if any) that match the criteria 4. The Controller forwards the resulting list of <code>bookmarks[]</code> to the View		3. The <i>FavoritesCollection</i> filters its bookmarks and returns only those matching the criteria to the Controller.		Figure 5.11-5
F_SR-3 Structure Favorites List: Set Ordering Criteria	1. The View listens for calls from the user to <code>setOrderingCriteria()</code> over the list of bookmarks. It forwards said call to the Controller	2. The Controller, in turn, orders the <i>FavoritesCollection</i> to sort itself accordingly		3. The <i>FavoritesCollection</i> sets the new ordering criteria as 'active' and uses it to sort the list of bookmarks		Figure 5.11-6
F_SR-4 Structure Favorites List: Group Bookmark(s)	1. The View listens for calls from the user to <code>addBookmarkToGroup()</code> . It forwards said call to the Controller	2. The Controller then orders <code>newGroup</code> to add the bookmark unto itself, but only after ordering said bookmark to be removed from <code>oldGroup</code>			3. <code>oldGroup</code> removes the <i>Bookmark</i> and <code>newGroup</code> adds it	Figure 5.11-7
F_SR-5 Structure Favorites List: Tag Bookmarks	1. The View listens for calls from the user to <code>addTagsToBookmark()</code> . It forwards said call to the Controller	2. The Controller orders the <i>Bookmark</i> to add the tags unto itself	3. <i>Bookmark</i> adds the <code>tags[]</code>			Figure 5.11-8

### 5.11.2.3 Usability Software Design Meta-models

The UML diagrams below represent the Low-level Design Component Responsibilities described in earlier. The following sections describe the class diagram and the classes and interrelationships involved in this feature, followed by the sequence diagrams.

#### 5.11.2.3.1 Class Diagram

Figure 5.11-3 below shows the class diagram for the Favorites Functional Usability Feature. As described in the Low-level Design Component Responsibilities Table (see Table 5.2-6), the main objects involved are the View, Controller, Bookmark, Tag, Group, FavoritesCollection. The first two, fulfilling their role within MVC, respectively capture and distribute the user calls to perform actions. The FavoritesCollection singleton is the gateway for accessing bookmarks and sorting them as requested, whether they belong to groups or not.

The Group class holds one or more Bookmarks, all which may be tagged by one or more Tags. Each bookmark holds an array of strings as part of their identification information as well as the list of tags associated to it.

Bookmarks are responsible for tagging themselves as requested.

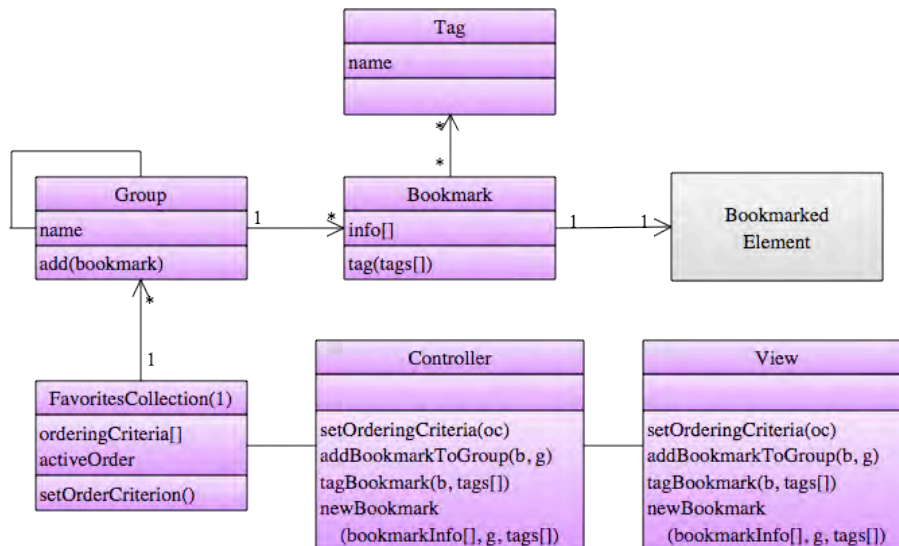


Figure 5.11-3: Usability Design Meta-model. Class diagram. Favorites.

#### 5.11.2.3.2 Sequence Diagram “Add Bookmark”

Figure 5.11-4 shows the sequence diagram for the Low-level Design Component Responsibilities of F\_SR-1 as described in Table 5.2-6. This diagram covers all the object responsibilities associated with creating a new bookmark and storing it. It starts with the user’s request to create a new bookmark. The view captures all the bookmark information along with that of any groups it is to belong to or tags it must hold. It passes this information to the Controller which forwards it to the FavoritesCollection.

The FavoritesCollection is the one responsible for creating the new bookmark with the provided information, tagging it (if applicable), finding the appropriate group and adding it to it (if applicable). If no group was selected upon creation of the bookmark, FavoritesCollection adds it to the defaultGroup, an instance of Group holding such ‘orphan’ bookmarks.

Classes and methods depicted in purple represent they belong to the Favorites feature. For the full color legend see page 71.

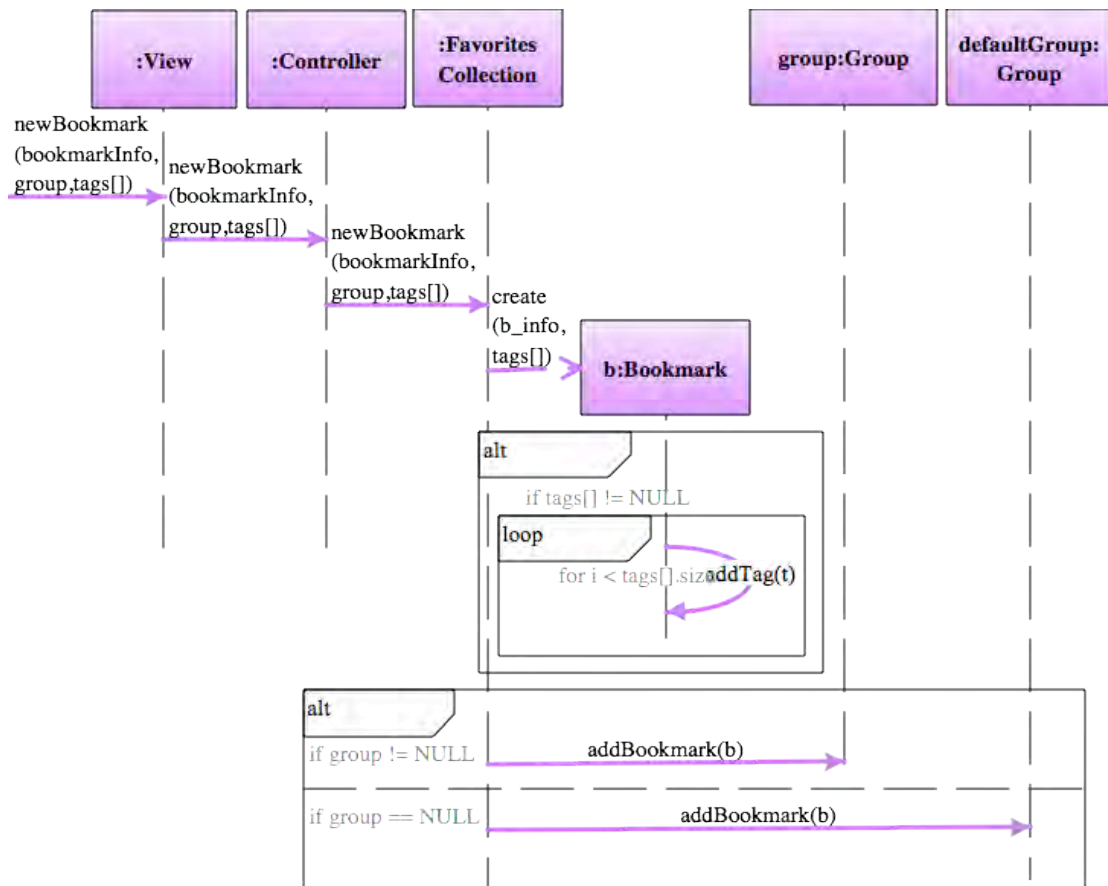


Figure 5.11-4: Sequence Diagram "Add Bookmark". Favorites.

### 5.11.2.3.3 Sequence Diagram "Retrieve Bookmark"

Figure 5.11-5 shows the sequence diagram for F\_SR-2 This diagram covers all the object responsibilities related to retrieving a bookmark.

This sequence starts when the user selects a previously stored Bookmark from the Favorites List. The View receives this request and passes it on to the Controller, which in turn requests the FavoriteCollection to return the Bookmark matching the request.

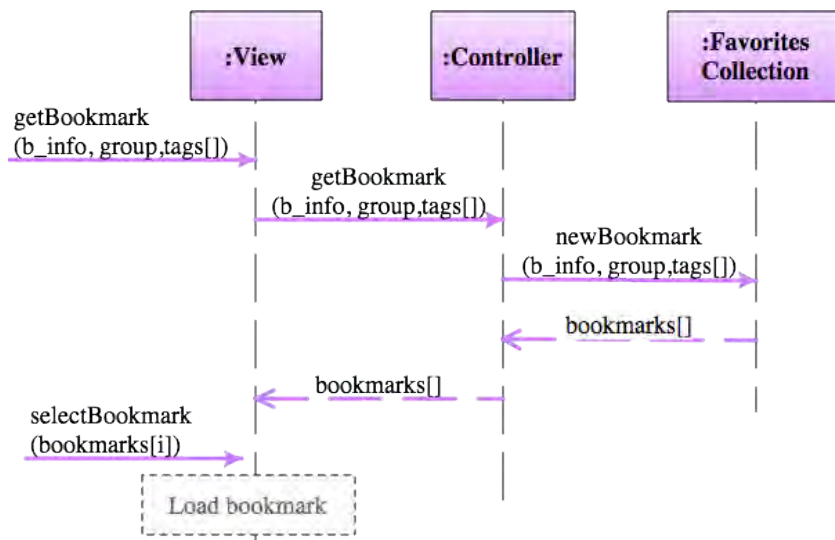


Figure 5.11-5: Sequence Diagram "Retrieve Bookmark". Favorites.

#### 5.11.2.3.4 Sequence Diagram "Set Order"

Figure 5.11-6 shows the sequence diagram for F\_SR-1, covering all the object responsibilities listed for sorting the Favorites List. It starts when the user sets a new order criteria by, for example, clicking on the header of a column where the bookmarks are shown (i.e. clicking on the 'date' column would order the bookmarks by date added). This call is forwarded via the Controller to the FavoritesCollection, which sets the new criteria (c) as the 'active criteria'.

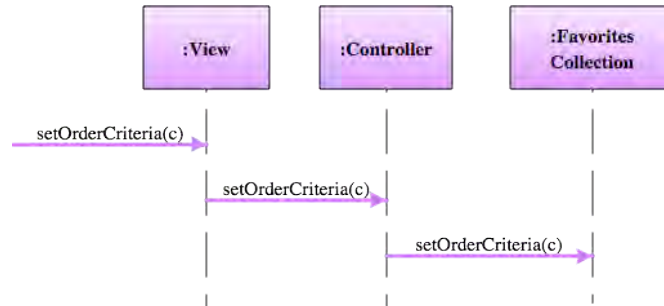


Figure 5.11-6: Sequence Diagram "Set Order". Favorites.

#### 5.11.2.3.5 Sequence Diagram "Add Bookmark to Group"

Figure 5.11-7 shows the sequence diagram for F\_SR-4. This diagram covers all the object responsibilities listed for adding a bookmark to a group. The sequence starts when the user requests to add an existing bookmark to a new group. The View gathers this information along with the group currently holding the bookmark to move (this would be the defaultGroup for unsorted bookmarks). This call is forwarded to the Controller which finds the current group, removes the bookmarks, finds the new group and adds it to it.

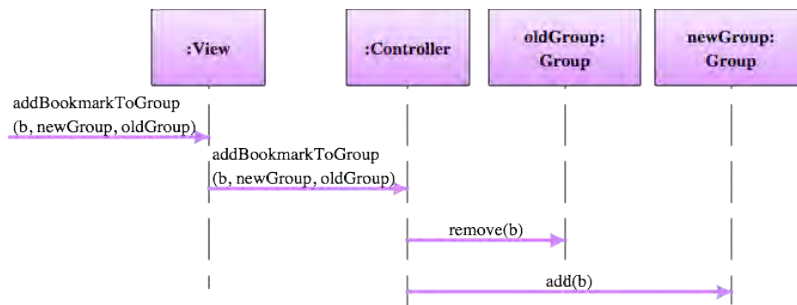


Figure 5.11-7: Sequence Diagram "Add Bookmark to Group". Favorites.

#### 5.11.2.3.6 Sequence Diagram "Tag Bookmark"

Figure 5.11-8 shows the sequence diagram for F\_SR-5. This diagram covers all the object responsibilities listed for tagging a bookmark. The sequence starts when the user requests to tag an existing bookmark. The View captures this information, including the tag(s) the user may want to add to the bookmark. This call is forwarded to the Controller which finds the bookmark in question and orders it to tag itself with each of these tags.

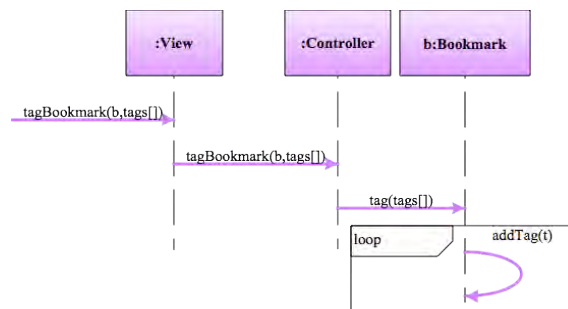


Figure 5.11-8: Sequence Diagram "Tag Bookmark". Favorites.

## 5.12 “Personal Object Space” Usability Guideline for Software Development

The Personal Object Space Functional Usability Feature covers allowing the user to arrange and manipulate objects graphically on screen. Users may need to lay out and organize application elements to his convenience and not be bound by pre-determined setups that may restrict his performance

The Usability Guideline for Software Development is made up of Analysis and Design artifacts, described for the Warning feature in the following two sections.

### 5.12.1 Usability Guideline for Software Development: Analysis artifacts

There are four artifacts to be used during the analysis phase: the Usability Elicitation Guideline, the Usability Elicitation Clusters, the Usability Use Case Meta-models, the System Responsibilities for Usability. These artifacts are described in the following four sections.

#### 5.12.1.1 Usability Elicitation Guideline

Table 5.6-1 shows the Usability Elicitation Guideline for the Undo Functional Usability Feature. In this guideline, there is a single HCI recommendation, described below.

##### 5.12.1.1.1 Personal object space

HCI authors suggest that users should be provided with a canvas where they could arrange things (i.e. graphic representations of application functionality) to their liking. (POS\_HCI-1). This is most commonly achieved by allowing the user to organize objects within a grid, though other presentations are possible (POS\_ELAB-1). During elicitation, discussions with stakeholders must determine which objects will be suitable for arranging in a personal object space (or a series of spaces), how they will be organized and information on default object positions and maximum number of objects allowed. (U\_Q-1 to U\_Q-3)

In Table 5.6-1, example POS\_EX-1 “iGoogle’s POS” describes an example for this HCI recommendation.

Table 5.12-1. Usability Requirements Elicitation Guideline. Personal Object Space.

Identification			
Name	Personal Object Space		
Family	User Profile		
Aliases	Personal Object Space [42] Personalized 'MY' site [49]		
Intent			
Allowing the user to arrange and manipulate objects graphically on screen			
Problem			
User may need to layout and organize application elements to his convenience and not be bound by pre-determined setups that may restrict his performance.			
Context			
Applications for which a set of objects or functionality can be represented graphically in a canvas for the user to manipulate and use.			
Interrelationships			
Undo: Not considering the 'undo' feature will mean that any changes to a user's personal space (adding, moving, deleting objects) cannot be undone			
Warning: Not considering the 'warning' feature will mean that the system will not be able to alert the user about, for example, objects deletion (which, when not undoable, might need to display a warning)			
HCI Recommendation	Elaboration	Discussions with Stakeholders	Usage Examples (optional)
<p>POS_HCI-1 Personal object space</p> <p>The user should be able to arrange things in a way that works best for him, since he knows more about how he works than the artefact's designer does. This way he can better remember where things are than if the items are arranged for him [42]. Allow users to place things where they want, at least in one dimension but preferably in two. It is tedious for the user to do all the item placement themselves, especially if they want precision or a sorting order. So, start out with a reasonable default layout. However, permit stacking, moving, grouping, aligning, "neatness" adjustments, sorting and other layout operations. Do not capriciously rearrange the user's space, only do automatic layout if the user specifically requests it. The artifact should maintain the user's layout between users.</p>	<p>POS_ELAB-1: Personal object space: layout</p> <p>The Personal Object Space is most commonly achieved by using a grid in which to place objects, which can be arranged by column, row, or both.</p>	<p>POS_Q-1 Will users be allowed to align certain objects graphically in a personal space?</p> <p>POS_Q-2 If so, which objects?</p> <p>POS_Q-3 What is the maximum number of objects to lay out?</p> <p>POS_Q-4 How will objects be arranged on screen?</p> <p>POS_Q-5 How will users move objects within layout?</p> <p>POS_Q-6 When an object is deleted, what happens to remaining objects?</p> <p>POS_Q-7 Where will a newly created object fall within arrangement?</p> <p>POS_Q-8 What will be the default arrangement?</p> <p>POS_Q-9 Will user be allowed to arrange objects within more than one space?</p> <p>POS_Q-10 How will users navigate between spaces?</p> <p>POS_Q-11 What is the maximum number of spaces?</p>	<p>POS_EX-1 iGoogle's POS</p> <p>In iGoogle users are allowed to place 'widgets' in their own personal object space. Widgets are laid out in a grid, and distributed by columns.</p> <p>Deleting a widget shifts the rest of the widgets in that column up.</p> <p>Every new widget falls in the (1,1) position of the grid</p> <p>Multiple spaces are allowed and they are represented by tabs to allow navigation</p>

### 5.12.1.2 Usability Elicitation Clusters

The Usability Elicitation Guideline suggests eleven discussions items to be held with stakeholders in order to elicit all aspects of the Personal Object Space Functional Usability Feature. These discussion items can be clearly divided into three initial groups, or clusters, as described in the Usability Elicitation Clusters, shown in Figure 5.12-1 according to the portion of the Personal Object Space functionality that they cover.

**POS\_EC-1 Arranging objects within space:** The discussion items in this cluster cover determining which objects will be suited for graphic arrangement on a Personal Object Space (POS), the maximum number of objects that may be allowed, how the objects will be arranged on screen and how they will be moved around a repositioned.

**POS\_EC-2 Handling default positions:** This cluster deals with the state of a space after object manipulation. Discussions entail determining what will happen to the objects in a space once one is deleted, or a new one is added or moved. Stakeholders must also determine what the default arrangement for a space will be, prior to user manipulation.

**POS\_EC-3 Managing multiple spaces:** This last cluster contains discussion items regarding the need for having more than once space for laying out objects. Should this need be present for a system, stakeholders must determine how navigation among spaces will take place and the maximum number of spaces that will be allowed.



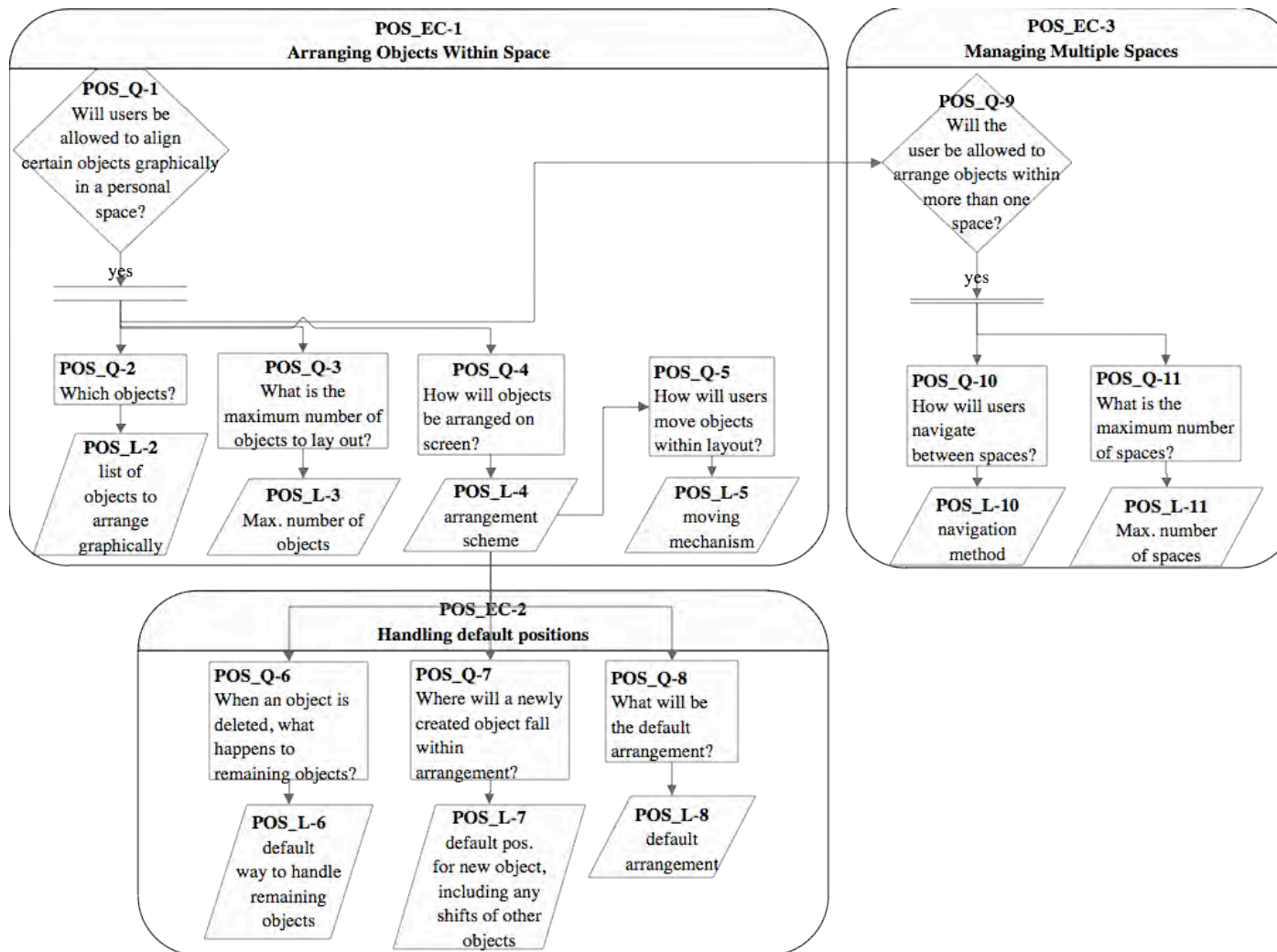


Figure 5.12-1: Elicitation Clusters. Personal Object Space.

### 5.12.1.3 Use Case Meta-model

The Use Case Meta-model for the Personal Object Space feature is shown in Figure 5.12-2(See page 71 for color legend), in which six use cases are identified and described below.

**POS\_UC-1 ManageObject:** This is the parent use case for moving an object (POS\_UC\_3), adding an object (POS\_UC\_4) and deleting an object (POS\_UC\_5) from the personal object space. These use cases all have in common the fact that, after each of their respective operations have been carried out (moving, adding, deleting) in all cases, the space in which they were performed must be rearranged around the change, as explained below.

**POS\_UC-2 RearrangeSpace:** Rearranging a space entails moving the space's existing objects so that they conform to a set of rules. Rearrangement only happens after an object in the space has been 'managed' in some way (see use case above), possibly leaving the space in an unstable state, needing to be rearranged for that stability to be attained once again. For example, if a new object is deleted from a space, other objects in that space may need to be repositioned to fill this newly vacated spot. Similar scenarios play out when moving or adding objects

**POS\_UC-3 MoveObject:** This use case starts when the user picks up an object from the space and moves it to a new position. This new position may be occupied or empty. After the object is moved, the remaining elements of the space are rearranged as/if needed

**POS\_UC-4 AddObject:** This use case starts when the user selects the option to add a new object to the space. The object can be added to a specific position of the space (occupied or not) or to the 'default' position in the space to which new objects are added.

**POS\_UC-5 DeleteObject.** Deletion starts when the user selects the object he wishes to delete and then chooses the option to eliminate it from the space. The rest of the objects in the space may have to be rearranged around the deletion (i.e. to fill out the newly vacated position)

**POS\_UC-6 GoToSpace:** When multiple spaces are allowed, this use case describes how the user can switch from one to another

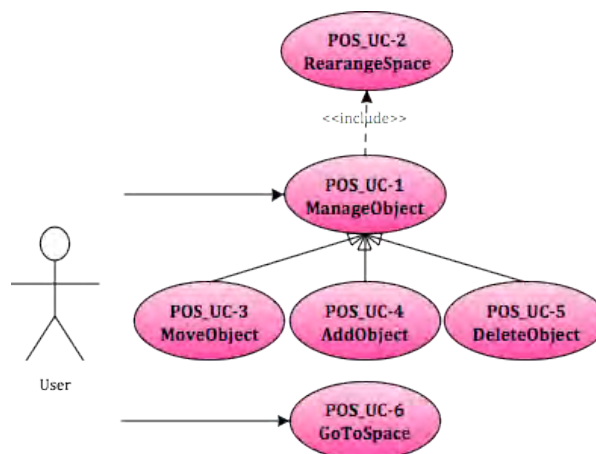


Figure 5.12-2 Use Case Meta-model. Personal Object Space

As mentioned above, the applicability of each of these use cases will depend on the results of the elicitation process. If during elicitation of the Personal Object Space Functional Usability Feature it is determined that, for example, multiple spaces will not be needed, then the GoToSpace (POS\_UC-2) use case would be discarded. Use cases also depend on one another. These dependencies are shown in Table 5.12-2, where we can see the following:

- The **ManageObject** use case and all of its children need the **RearrangeSpace** use case, because every time an object is deleted/moved/added within a space, the rest of the objects will most likely need to be rearranged to leave the space in a stable state.
- The **RearrangeSpace** is included within the **ManageObject** use cases, and needs them to exist, because the user never calls it directly, only through these use cases.
- The **GoToSpace** use case needs no other use case to be viable, as it is only a means to move among spaces, regardless of the activities that may take place in them.

Table 5.12-2 Usability Use Case Dependencies: Personal Object Space Functional Usability Feature

	POS_UC-1 Manage Object	POS_UC-2 Rearrange Space	POS_UC-3 MoveObject	POS_UC-4 AddObject	POS_UC-5 DeleteObject	POS_UC-6 GoToSpace
POS_UC-1 ManageObject	-	X				
POS_UC-2 RearrangeSpace	X	-				
POS_UC-3 MoveObject	X	X	-			
POS_UC-4 AddObject	X	X		-		
POS_UC-5 DeleteObject	X	X			-	
POS_UC-6 GoToSpace						-

By looking at the columns in Table 5.12-2, it becomes evident that, aside from ManageObject, RearrangeSpace is core to the Personal Object Space feature.

#### 5.12.1.4 System Responsibilities

Table 5.12-3 shows the proposed System Responsibilities for Usability for the present feature.

Table 5.12-3 System Responsibilities List for Personal Object Space

System Responsibilities List for Undo
POS_SR-1 Initialize space The system must know how objects are represented on screen and the rules that govern a space
POS_SR-2 Move Object The system must allow for objects to be moved within a space
POS_SR-3 Add Object The system must allow for new objects to be added to a space
POS_SR-4 Delete Object The system must allow the user to delete objects from a space
POS_SR-5 Manage multiple spaces The system must allow the user to navigate from one space to another

These System Responsibilities are derived from the Elicitation Clusters as explained below:

**POS\_EC-1 Arranging objects within a space:** This cluster contains the discussions that determine the ‘rules’ of every space (which objects are allowed, how they’ll be arranged, etc.). It also deals with the rules governing object movement within a space. As such, this elicitation cluster would yield two System Responsibilities, namely **POS\_SR-1 Initialize space** and **POS\_SR-2 Move object**.

**POS\_EC-2 Handling default positions:** This cluster deals with the default positions objects will go to when added to a space. Also, it details how after deletion/addition of a new object into a space, said space will regain stability (i.e. once again conform to the rules mentioned above). This cluster then gives way to two more system responsibilities: **POS\_SR-3 Add object** and **POS\_SR-4 Delete object**.

**POS\_EC-3 Managing multiple spaces:** This cluster focuses on handling multiple spaces in a system, producing one System Responsibility **POS\_SR-5 Move object**.

Table 5.12-4 maps the relationships between the Usability Elicitation Clusters and the Usability System Responsibilities described above, for easy reference. Any project determined to require a specific Elicitation Cluster will also require its related System Responsibilities. Likewise, if a cluster is discarded during elicitation, its related responsibilities will not be a part of the resulting system.

Table 5.12-4 Usability Elicitation Clusters / System Responsibilities Mapping for Personal Object Space

Elicitation Clusters	Dependent Responsibilities
POS_EC-1 Arranging objects within a space	POS_SR-1 Initialize Space POS_SR-2 Move Object
POS_EC-2 Handling default positions	POS_SR-3 Add Object POS_SR-4 Delete Object
POS_EC-3 Managing multiple spaces	POS_SR-5 Move Object

## 5.12.2 Usability Guideline for Software Development: Design artifacts

The design artifacts of the Usability Guideline for Software Development for the Warning feature are described in the following sections. The System Responsibilities described above are brought to a lower abstraction level as High-level Design Component Responsibilities in section 5.12.2.1. Section 5.12.2.2 expresses them as Low-level Design Component Responsibilities (for a MVC architecture). Finally, section 5.12.2.3 presents the Usability Design Meta-models for said Low-level Design Component Responsibilities as object-oriented class and sequence diagrams.

### 5.12.2.1 High-level Design Component Responsibilities

In order to support the System Responsibilities at design level, the following sections describe the suggested High Level Design Components for the Progress feature, shown in Table 5.2-5.

#### 5.12.2.1.1 User Interface (UI) Component

This component is responsible for capturing all user invocations and forwarding them (possibly through a delegating component) to the appropriate part of the domain, usually that responsible for executing the invoked action. For example, every time the user drags-and-drops an object in a space, the UI must be aware of the move, and forward it to the part of the domain responsible for determining what the move means

#### 5.12.2.1.2 Domain Component

A Domain Component represents the part of the system functionality that is represented graphically on-screen as an *object*.

For example, in an application like iGoogle, the Domain Component is the *widget* (widgets hold application functionality). An iGoogle space is made up of a grid of widgets that the user can organize as he best sees fit.

#### 5.12.2.1.3 Placeholder Component

A Placeholder Component represents an unmovable place within a space, with a defined position. It is responsible for storing a single object (Domain Component) within a space.

#### 5.12.2.1.4 Space Component

A space is represented by the Space Component. The Space Component is responsible for rearranging objects upon request from the UI (according to a positioning scheme, see below) whenever a new object is added, deleted or moved.

#### 5.12.2.1.5 Home Component

A Home Component is responsible for holding all Space Components. It is responsible for knowing the order in which spaces are laid out. It is also responsible for knowing which is the

space to be loaded upon application startup, and whether more (or even all) spaces will need to be loaded in the background when doing so.

#### 5.12.2.1.6 Positioning Scheme Component

This component is responsible for knowing the rules pertaining to how objects are laid out within a space. Given any (continuous) point in the UI, the Positioning Scheme Component is responsible for determining what Position (see component below) it represents. It is also responsible for knowing where the next empty position is, depending on its internal rules of order (by column, by row, etc)

#### 5.12.2.1.7 Position Component

A Position Component represents a place where a Placeholder Component can reside within a Space Component.

Table 5.12-5: Usability Guideline: High-level Design Component Responsibilities. Personal Object Space

System Responsibility	Generic Component Responsibilities
POS_SR-1 Initialize space	A Space component is responsible for loading itself upon request. The Home component is responsible for knowing which space should be loaded at each time
POS_SR-2 Move Object	A Placeholder Component represents an unmovable place within a space, with a defined position, where a single object can be stored. A Placeholder Component lives within a Space Component, which represents a single user space. The Space Component is responsible for rearranging objects upon request from the UI, according to a positioning scheme, whenever a new object is added, deleted or moved. A Positioning Scheme Component holds the rules pertaining to how objects are laid out within a space. Given any (continuous) point in the UI, the Positioning Scheme Component is responsible for determining a single Position. It is also responsible for knowing where the next empty position is, depending on its internal rules of order (by column, by row, etc) A Position Component represents a place where a Placeholder Component can reside within a Space Component.
POS_SR-3 Add Object	The Space Component is also responsible for knowing a set of default positions for its Placeholders, should the user request to reset to one.
POS_SR-4 Delete Object	A Placeholder Component where an object resides is responsible for deleting it upon request by the Space Component that contains it.
POS_SR-5 Manage multiple spaces	A Home Component is responsible for holding multiple Space Components when needed. It is responsible for knowing the order in which spaces are laid out (and which one is the first to be loaded upon application start)

#### 5.12.2.2 Low-level Design Component Responsibilities for MVC

When instantiating for a MVC architecture, the UI Component is represented by View object(s) and takes over all of its responsibilities except for the delegation of actions to other objects. This responsibility falls on the Controller object(s) of the MVC architecture.

The components in the previous section are represented by the objects of the same name in Table 5.2-6, and each all the responsibilities described for their respective components described in the previous section

Table 5.2-6 details the Low-level Design Component Responsibilities and how they carry out each of the System Responsibilities defined in section 5.2.1.4. For each System Responsibility, the sequence of actions required by the different objects is presented as well as a set of UML diagrams that depict each of these interactions.

Table 5.12-6: Usability Guideline: Low-level Design Component Responsibilities (MVC). Personal Object Space.

System Responsibility	Objects						Fig
	View	Controller	Space	PositioningScheme	Home	Placeholder	
POS_SR-1 Initialize space	1. The View listens for the user call to loadStartSpace() and forwards it to the Controller. 7. The View displays the first space of those received from the Controller. If only one was received, it is the start space itself, and is loaded.	2. The Controller orders the Home to load the initial Space(s) 6. The Controller forwards the loaded spaces to the View.	4. Each Space loads itself (loads all the objects within it, getting ready for display)		3. The Home, knowing which Spaces need to be loaded together with the start space (if applicable) orders it/them to load 5. The Home returns the loaded spaces to the Controller.		Figure 5.12-4
POS_SR-2 Move Object	1. The View listens for user call to move() an object from one point in the screen (from_placement_data[]) to another (to_placement_data[]), and forwards the call to the Controller, along with the identifier of the current active_space 8. The View updates accordingly after being notified of changes in the current Space	2. The Controller finds the current active_space and orders it to move the object according to the from/to data passed on by the View	3. The Space, calls unto its PositioningScheme to determine what Position corresponds to the to_placement_data[] sent along by the View. 5. Space finds the Placeholder corresponding to that Position, gets() its current object, if any, and puts() the new object inside. 7. Space rearranges() its remaining objects according to its internal rules (i.e. swapping, shifting) and notifies the View	4. The PositioningScheme determines the Position corresponding to the screen coordinates and returns it to the Space		6. Placeholder substitutes its current object by the new object	Figure 5.12-5
POS_SR-3 Add Object	1. The View listens for user call to add() an object at to the current space, and forwards the call to the Controller, along with the identifier of the current active_space 8. The View updates accordingly after being notified of changes in the current Space	2. The Controller finds the current active_space and orders it to add the object to itself	3. The Space, calls unto its PositioningScheme to determine the default Position to add new objects 5. Space finds the Placeholder corresponding to that Position and puts() the new object inside. 7. Space rearranges() its remaining objects according to its internal rules and notifies the View	4. The PositioningScheme determines the Position corresponding to the screen coordinates and returns it to the Space		6. Placeholder adds the new object	Figure 5.12-6
POS_SR-4 Delete Object	1. The View listens for user call to delete() an object in a specific location in the screen (placement_data[]), and forwards the call to the Controller, along with the identifier of the current active_space 8. The View updates accordingly after being notified of changes in the current Space	2. The Controller finds the current active_space and orders it to delete the object according in that placement	3. The Space, calls unto its PositioningScheme to determine what Position corresponds to the placement_data[] sent along by the View. 5. Space finds the Placeholder corresponding to that Position and deletes() its current object 7. Space rearranges() its remaining objects according to its internal rules (i.e. swapping, shifting) and notifies the View	4. The PositioningScheme determines the Position corresponding to the screen coordinates and returns it to the Space		6. Placeholder deletes its current object	Figure 5.12-7
POS_SR-5 Manage multiple spaces	1. The View listens for user calls to goToSpace(s) 2a. If the system is set up to load all spaces upon startup (and, therefore, s and all its elements are already loaded) the View proceeds to makeActive(s) and display(s) 2b. If the system only loads an initial space upon startup, the request for space s is forwarded to the Controller. 5. The View loads the space	2. The Controller forwards the request to the Home object 4. The Controller receives the space and passes it on to the view			3. Home finds the Space with id space_id and asks it to load() all of its components. Home then returns space to the Controller		Figure 5.12-8

### 5.12.2.3 Usability Software Design Meta-models

This section describes the UML diagrams representing the Low-level Design Component Responsibilities described above. Below, the class diagram is described, as well as the classes involved in this feature and their interrelationships, followed by the descriptions of the sequence diagram.

#### 5.12.2.3.1 Class Diagram

Figure 5.2-3 below shows the class diagram for the Personal Object Space feature. The main objects involved are the View, Controller, Home, Space, Placeholder, PositioningScheme, Position. Additional classes such as Grid, GridPosition, Rows, RowsPosition, etc., are also present in the class diagram and are explained below.

The first two classes, View and Controller fulfilling their role within MVC, respectively capture and distribute the user calls to perform actions. The Home class holds multiple spaces (or a single one, in systems where only one space is needed). The Space class holds many Placeholders, and each Placeholder is the location, with a given Position, where a DomainClass is stored.

Every space has a single PositioningScheme, which is a set of rules, which govern the Space's layout and ordering. The Positioning Scheme can be a Grid, Rows, Columns, Absolute (positioning) and so on. These concrete schemes are depicted in the class diagram, but many more can exist. The rules in each concrete scheme will be unique, and will define the scheme. Such rules could include, for example, that any new object added to a Space governed by a Columns positioning scheme will always fall at the bottom of the last row.

Finally, the Position object, associated with every Placeholder in a Space, also has concrete inheriting classes, closely related to the chosen PositioningScheme. For example, in a Space ordered by Rows, a Position will be of the type RowsPosition, and may read "object is in row 7, position 3". A position for a Grid scheme may read "object is in position (3,4)" while an Absolute scheme will lead to Absolute Positions, of the type "h: 723px, w: 123px".

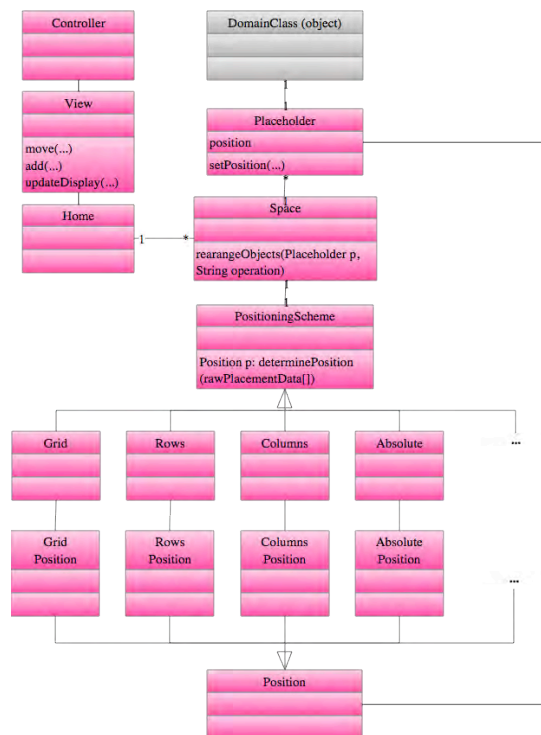


Figure 5.12-3: Usability Design Meta-model. Class diagram. Personal Object Space.

Classes and methods depicted in dark pink represent they belong to the Personal Object Space fdature. The gray DomainClass is a template class to be substituted at desing time by the system class containing the undoable action. For the color legend see page 71.

#### 5.12.2.3.2 Sequence Diagram "Load Start Space"

Figure 5.12-4 shows the sequence diagram for the Low-level Responsibilities corresponding to the System Responsibility POS\_SR-1 Initialize Space.

This sequence starts when the user requests for the initial space to be loaded. This usually happens automatically at application startup, but can be called independently for flexibility. The call is captured by the View and forwarded to the appropriate Controller, which locates the Home comopnent and asks it for its spaces.

Depending on what has been elicited, starting up an application may load more than just the start space. For example, some systems may require all spaces to be loaded at startup, while just displaying the first. In any case, it is the Home's responsibility to know which spaces (other than the start space, if any) are to be loaded upon startup and returns the set to the Controller. This is forwarded to the View, which picks the first element of the array (the start space) and displays it for the user, setting it as the active space

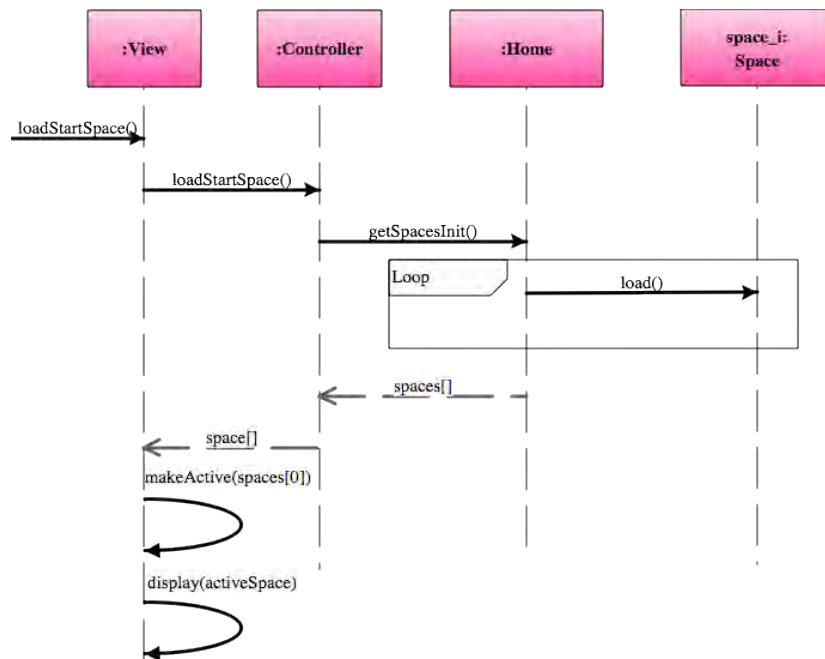


Figure 5.12-4: Sequence Diagram "Initialize Space". Personal Object Space.

#### 5.12.2.3.3 Sequence Diagram "Move Object"

Figure 5.12-5 shows the sequence diagram for the Low-level Design Component Responsibilities corresponding to the System Responsibility POS\_SR-2 Move Object.

This sequence starts when the user moves an object from one point in the screen to another. The View captures the raw value of the exact point of origin (from\_placement\_data[]) and the exact point where the user released the mouse button (to\_placement\_data[]). This information is sent to the Controller who locates the current active spaces and orders it to move the selected object and gives it the too raw screen coordinates.

The active space calls unto its Positioning Scheme and orders it to calculate the Position (object) for that raw screen data, corresponding to the point of destination (where the object was 'dropped' by the user). The Positioning Scheme calculates the Position and returns it to



the active Space, which locates the Placeholder at that Position. If an object is currently residing in that Placeholder, it is taken out (held) and the new object is moved in.

The state of the space is now the following: The original placeholder for the object being moved is empty, the destination placeholder contains the moved object, any object that was there previously is on hold, and the space still needs to be returned to a stable state.

The following step is for the Space to order the PositioningScheme to calculate the Position for the original Placeholder. With this information, the Space is now capable of rearranging itself according to its internal rules (including returning to a state of stability, and placing the held object where appropriate).

Once the space is done rearranging itself, it notifies the View, which updates the display to reflect the changes.

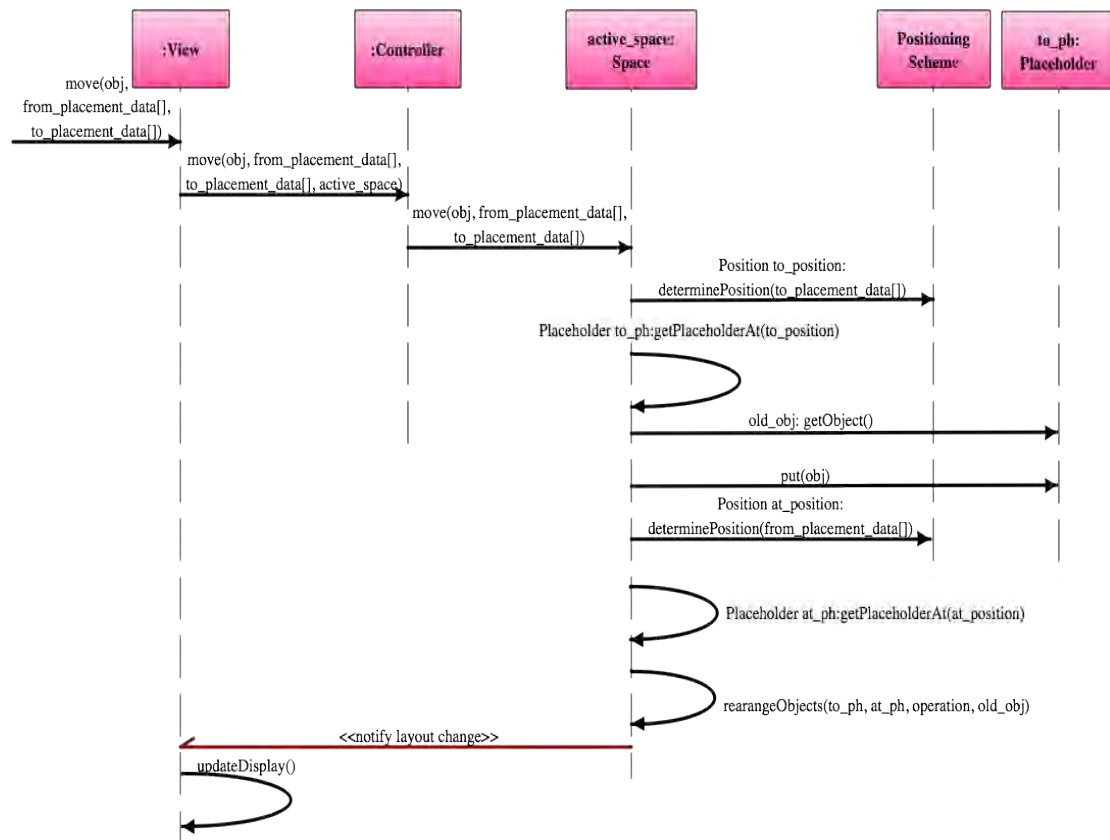


Figure 5.12-5: Sequence Diagram "Move Object". Personal Object Space

#### 5.12.2.3.4 Sequence Diagram "Add Object"

Figure 5.12-6 shows the sequence diagram for the Low-level Design Component Responsibilities corresponding to the System Responsibility POS\_SR-3 Add Object.

This sequence starts when the user selects the option to add a new object to the space. The View captures the call and forwards it, along with the object and the id of the space to which it's being added, to the Controller. The Controller finds the space by its id and orders it to add the object into itself.

Being given no specific location where to add the object, the Space determines its default Placeholder and adds the object to it, rearranging the rest of its objects accordingly. Once that is done, the Space notifies the View, which updates itself to reflect the changes.

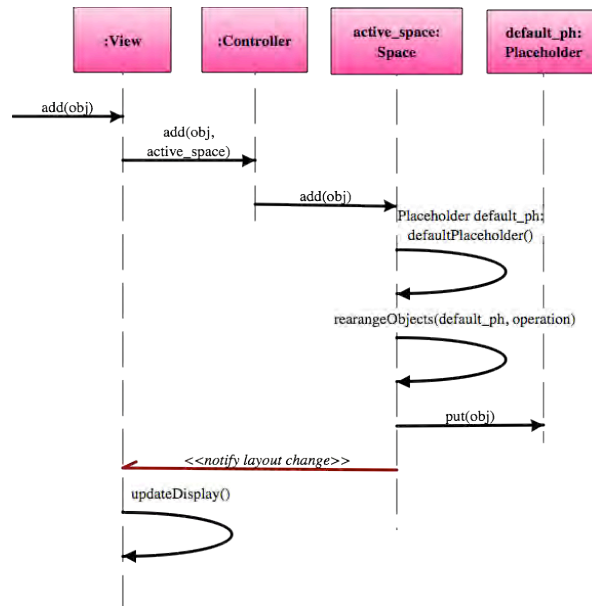


Figure 5.12-6: Sequence Diagram "Add Object". Personal Object Space.

#### 5.12.2.3.5 Sequence Diagram "Delete Object"

Figure 5.12-7 shows the sequence diagram for the Low-level Design Component Responsibilities corresponding to the System Responsibility POS\_SR-4 Delete Object.

This sequence starts when the user chooses an object within a space and selects the option to delete it. The View captures this call and, along with the id of the current active space, the object being deleted and its location (placement\_data) forwards it to the controller. The controller finds the Space and orders it to delete the object located at that placement\_data. With this information, the space first orders the PositioningScheme to calculate the Position corresponding to that placement\_data. With that Position, the Space can locate the Placeholder that is being affected, and order it to empty itself.

After emptying the Placeholder, the Space rearranges itself around the change and notifies the View once its done. The View, updates itself to reflect the new state of the Space.

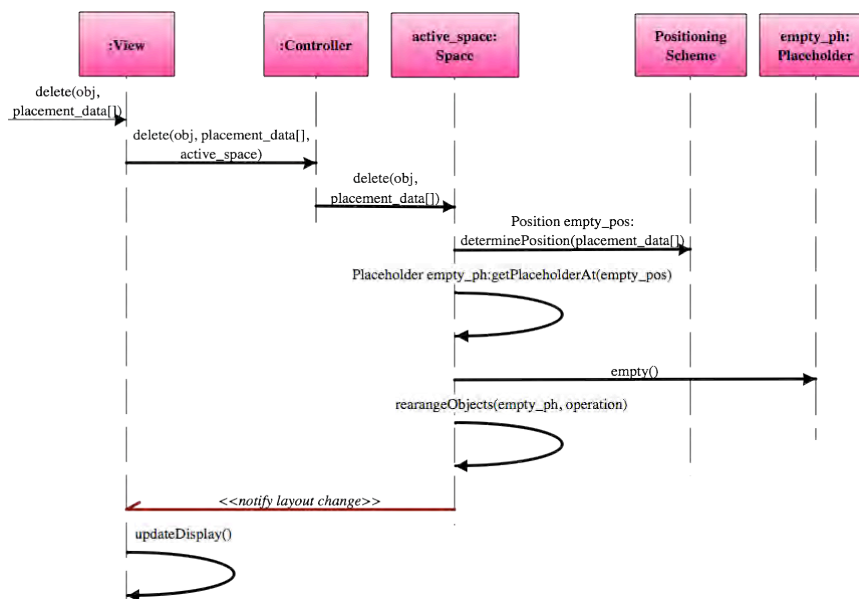


Figure 5.12-7: Sequence Diagram "Delete Object". Personal Object Space.

### 5.12.2.3.6 Sequence Diagram "Go to space"

Figure 5.12-8 shows the sequence diagram for the Low-level Design Component Responsibilities corresponding to the System Responsibility POS\_SR-5 Manage Multiple Spaces.

This sequence begins when the user chooses to go to a different space from the one he's currently in. Two possibilities arise, depending on whether or not all spaces were loaded onto memory during application start up. If they were (else portion of the alt box in Figure 5.12-7), the View has direct access to them, and simply makes the space selected by the user be the new active space, and displays it.

If during application startup only the initial space was loaded, then when switching to a new space, the system must locate that space first. The call to load the new space is forwarded to the Controller who asks the Home class to locate the space corresponding to the given id. The Home locates the Space and orders it to load all of its objects, while returning it to the View through the Controller.

Having access now to the new Space, the View makes it the active space and displays it on screen, along with all of its objects.

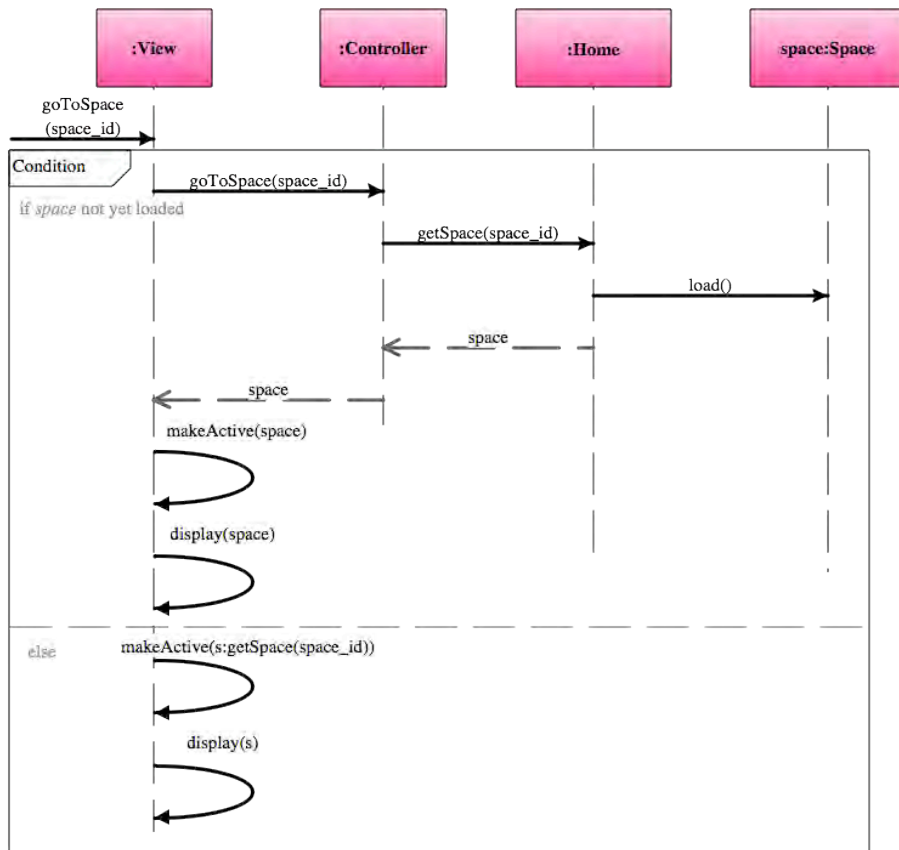


Figure 5.12-8: Sequence Diagram "Go to space". Personal Object Space.

# CHAPTER 6. VALIDATION

## 6.1 Introduction

Our proposed Usability-oriented Software Development Process and the Usability Guidelines for Software Development were tested across multiple projects to prove the proposed hypotheses regarding their usefulness. This chapter presents the experiments that were conducted and an analysis of their results, using an adaptation of the Common Industry Format (CIF) for Usability Test Reports [29].

## 6.2 Hypothesis

As described in Chapter 3, the hypothesis for this work proposes that:

“Applying the proposed usability-oriented software development process will:

- Reduce development **time** of the usability-related functionalities and, as a consequence, of the project over all,
  - improve the **quality** of resulting software designs,
  - facilitate the inclusion of functional usability features into software projects by reducing the **perceived complexity** of usability features by developers,
- over applying the process partially *and* over not applying it altogether.”

This general hypothesis is further broken down as follows for accurate validation:

“Applying the proposed usability-oriented software development process in full will:

- H1.** Reduce development **time** of the usability-related functionalities and, as a consequence, of the project over all, over applying the process partially,
- H2.** reduce development **time** of the usability-related functionalities and, as a consequence, of the project over all over not applying the process,
- H3.** improve the **quality** of resulting software designs, over applying the process partially,
- H4.** improve the **quality** of resulting software designs, over not applying the process,
- H5.** facilitate the inclusion of functional usability features into software projects by reducing the **perceived complexity** of usability features by developers, over applying the process partially,
- H6.** facilitate the inclusion of functional usability features into software projects by reducing the **perceived complexity** of usability features by developers, over not applying the process.

The proposed null hypothesis is the following:

“When applying the proposed usability-oriented software development process:

- development **time** is not reduced,
- **quality** of the resulting designs is not improved,
- **perceived complexity** of usability features is not reduced,

over applying the process partially *and* over not applying it altogether.”

which is similarly broken down as follows:

“When applying the proposed usability-oriented software development process:

- H1\_0.** Development time is not reduced over applying the process partially,
- H2\_0.** development time is not reduced over not applying the process,
- H3\_0.** quality of the resulting designs is not improved over applying the process partially,
- H4\_0.** quality of the resulting designs is not improved over not applying the process,
- H5\_0.** perceived complexity of usability features is not reduced over applying the process partially,
- H6\_0.** perceived complexity of usability features is not reduced over not applying the process,

Section 6.3 below describes the methods that were used to test the above hypotheses. Section 6.4 explains the variables that were observed and how data for them was collected. Section 6.5 describes the subjects who participated in the experiments, followed by the data that was collected, in Section 6.6. Section 6.7 elaborates the analysis of these data and finally, in section 6.8, the results and findings of the validation process as a whole.

### 6.3 Methods

To test the proposed hypotheses, nine subjects were assigned to develop the following software applications:

1. **An online task manager:** An application to manage to-do lists with the possibility of sharing and scheduling tasks, as well as organizing them visually.
2. **A console for a home automation system:** An application to operate a simulated network of sensors and actuators that controlled various features of a home environment (lights, air conditioner, blinds, garage door, etc.) in real time.
3. **An auction site:** A web application with basic auction functionalities.

These proposed projects were designed to be of similar complexities. See Appendix 9.4 for the full Software Requirement Specification (SRS) of each project.

The three SRSs for the three proposed projects already included usability requirements freeing subjects from the task of eliciting usability aspects in these experiments (first step of the proposed process), as the effectiveness of the elicitation guidelines has already been proven in [32].

With the focus on the remaining phases of the proposed process, each subject was expected to analyze, design and implement their software applications, including all usability aspects already present in the SRSs.

Each project was assigned to three different subjects, numbered PiSj (for Project ‘i’, Subject ‘j’). Each subject in each project was provided with the same SRS for that project as follows:

- Subjects numbered “1” for each project (the PiS1s), were asked to develop the project without any knowledge of the usability-oriented software development process.

- Subjects numbered “2” for each project (the PiS2s), were provided only with information regarding the elicitation and analysis phase of the usability-oriented software development process to develop their projects. In other words, they were kept from all information about the design phase of the process (including the software design-specific parts of the software usability guidelines).
- This would provide these subjects with clear guidance on how to perform the analysis of their project and an overview of the responsibilities that their system was expected to fulfill. They were, however, given no direction on how to design such responsibilities and were asked to do so to the best of their abilities.
- Subjects numbered “3” for each project (the PiS3s), were asked to develop their projects applying the full usability-oriented development process and, as such, were provided with the full software usability guidelines.

All teams developed their projects iteratively, covering the analysis, design, implementation and testing phases in each iteration. They produced analysis artifacts such as use case models and extended use cases, and design artifacts such as class and collaboration diagrams. Once finished, the results were analyzed as explained in the following sections. The results of each of these projects can be found on the digital media support that accompanies this work.

## 6.4 Variables

In the previous section we have established our independent variable to be “type of process used” in regards to our proposed process (i.e. full, partial or none). Furthermore, the following dependent variables were observed in each project in order to test the proposed hypotheses: development time, perceived complexity of the usability features and resulting design quality. Sections 6.4.1, 6.4.3 and 6.4.2 describe these metrics and how measurements were made in each case.

### 6.4.1 Development Time

Subjects were asked to closely measure the time they spent on the different phases of development. Specifically, they were asked to keep track of the amount of time they spent performing activities that pertained to any of the usability features.

These measurements would allow us to determine if a significant amount of time is saved during development of the usability-related functionality due to the application of the proposed process. Such a reduction would also imply, given that no other factors are altered, a reduction in the over-all development time for the project as a whole.

Development time is a ratio variable.

### 6.4.2 Resulting design quality

Software design quality can be defined by multiple factors, as described in [1][21] and [37]. The attributes that were chosen on which to evaluate projects, listed below, were selected from among the many proposed by [1] due to their relevance to our project and their ease of evaluation. Wide variations in these simple measurements among projects provide a general idea of the impact that the proposed process may have over design quality. The attributes are:

- Notation correctness
- Adequate responsibility allocation
- Diagram readability

After each project was finished, the resulting designs were evaluated by rating the aforementioned aspects on a 1 to 5 interval scale [47], where all pairs of adjacent scores are

equidistant. This means that a design scoring “5” is better than one scoring “4” in the same amount that one scoring “2” is better than another scoring “1”.

This evaluation was performed blindly by removing the name of the student (along with any other identifying information) from the designs, prior to evaluation.

### 6.4.3 Perceived complexity of usability mechanisms

In order to gauge the complexity of the usability mechanisms as perceived by the test subjects, they were asked to answer the following two questions on a 1-to-5 interval scale [47] upon completion of their projects:

“How would you rank this feature in terms of the complexity you encountered during the:

1. Design phase?”
2. Implementation phase?”

To reinforce the notion that all five possible marks are equidistant, this questionnaire was presented to subjects along with a visual analog scale, where the spacing between response levels was clearly indicated as equally spaced points in a line. Furthermore, they were presented with a brief description of the scale prior to filling out the questionnaire and an relatable example explaining that while a guideline with a complexity of “4” is not necessarily “twice as complex” as one with a complexity of “2”, the first is more complex than the second in the same amount as two guidelines scoring, for example, “5” and “3”.

Though the focus of this qualitative evaluation is design, a question about implementation was included in the questionnaire. This was done to determine whether designs produced when applying the proposed process helped developers beyond just that phase, transferring its benefits into the implementation phase as well by making the code that resulted from these designs seem less daunting to developers.

Answers to these questions would help to determine if there is a significant difference in how the features (and, by extension, usability in general within the scope of their projects) is perceived by developers, depending on whether they applied the proposed process in full, partially or not at all.

## 6.5 Subjects

The pool of subjects chosen to test the proposed hypotheses was comprised of final-year BS students as well as first-year master students of Software Engineering at the UPM School of Computing. As shown in Table 6.5-1, the subjects were chosen to have similar levels of knowledge in programming and software engineering, having attended the same school and taken most of the same courses related to these areas of interest. They were also chosen to have none to minimal work experience in the field.

Table 6.5-1 Characteristics and capabilities of the test subjects

Subject	Gender	UPM BS Prog. Courses	UPM BS SE Courses	UPM MSc SE Courses	Work Experience
P1S1(NP)	M	Yes	Yes	Yes	No
P1S2(PP)	F	Yes	Yes	No	No
P1S3(FP)	M	Yes	Yes	No	< 1 year
P2S1(NP)	F	Yes	Yes	Yes	No
P2S2(PP)	M	Yes	Yes	No	< 1 year
P2S3(FP)	M	Yes	Yes	No	No
P3S1(NP)	M	Yes	Yes	Yes	No
P3S2(PP)	M	Yes	Yes	No	2 years
P3S3(FP)	M	Yes	Yes	No	< 1 year

## 6.6 Data

The following sections present a synopsis of the data that was collected during experimentation for each of the variables presented in section 6.4

### 6.6.1 Development time data

Table 6.6-1 shows the average total times spent by subjects in developing the usability-related functionality for their assigned projects. Rows represent each development phase of the project, namely analysis, design, implementation and testing. Columns designate the degree to which the proposed process that was applied: full process (FP), partial process (PP) or none at all (NP).

Table 6.6-1 Average total time (in min) to develop all usability-related functionality

	NP	PP	FP
Analysis	250.00	215.33	247.33
Design	536.00	365.67	207.75
Implementation	771.67	579.67	334.71
Testing	408.67	206.67	23.67
<b>TOTAL</b>	<b>1,966.33</b>	<b>1,367.33</b>	<b>813.46</b>

The values above were obtained by averaging and adding up the times that the subjects spent, during each of these phases, in developing the software functionality related to each of the functional usability features. Table 6.6-2 to Table 6.6-5 show these source values. These values are the times spent by subjects during the design of the parts of their projects related to the usability features.

The first three columns in each section represent the project (P1, P2, P3), followed by the mean (avg) and standard deviation (stdv) for each type of process that was applied (NP, PP, FP). At the bottom of each table is the sum of these averages, shown above, in Table 6.6-1.

Table 6.6-2 Time (in min) spent by subjects in analyzing the usability-related functionalities of their project

	NP (P1)	NP (P2)	NP (P3)	NP avg	NP stdv	PP (P1)	PP (P2)	PP (P3)	PP avg	PP stdv	FP (P1)	FP (P2)	FP (P3)	FP avg	FP stdv
Abort	10	30	27	22.33	10.79	34	25	30	29.67	4.51	10	25	32	22.33	11.24
Undo	10	50	30	30.00	20.00	30	18	20	22.67	6.43	90	20	15	41.67	41.93
Comm	13	45	45	34.33	18.48	10	22	20	17.33	6.43	10	25	20	18.33	7.64
Prog	12	40	30	27.33	14.19	35	21	15	23.67	10.26	72	20	25	39.00	28.69
Status	13	15	12	13.33	1.53	19	25	18	20.67	3.79	15	20	15	16.67	2.89
Warn	10	20	39	23.00	14.73	5	18	25	16.00	10.15	30	15	20	21.67	7.64
Help	13	15	31	19.67	9.87	5	13	10	9.33	4.04	19	15	13	15.67	3.06
Fav	15	20	21	18.67	3.21	15	10	18	14.33	4.04	30	8	14	17.33	11.37
POS	17	30	20	22.33	6.81	10	25	30	21.67	10.41	19	15	28	20.67	6.66
Pref	13	18	28	19.67	7.64	5	15	35	18.33	15.28	13	12	32	19.00	11.27
SbS	18	20	20	19.33	1.15	20	25	20	21.67	2.89	18	5	22	15.00	8.89
<b>TOTAL</b>	250.0					215.3					247.3				

Table 6.6-3 Time (in min) spent by subjects in designing the usability-related functionalities of their project

	NP (P1)	NP (P2)	NP (P3)	NP avg	NP stdv	PP (P1)	PP (P2)	PP (P3)	PP avg	PP stdv	FP (P1)	FP (P2)	FP (P3)	FP avg	FP stdv
Abort	10	120	29	53.00	58.80	30	107	35	57.33	43.09	40	20	20	26.67	11.55
Undo	81	150	120	117.00	34.60	43	80	45	56.00	20.81	43	30	25	32.50	9.01
Comm	70	93	85	82.67	11.68	25	25	35	28.33	5.77	0	30	20	16.67	15.28
Prog	35	100	70	68.33	32.53	27	70	50	49.00	21.52	23	35	25	27.50	6.61
Status	22	35	7	21.33	14.01	23	28	31	27.33	4.04	13	25	25	20.83	7.22
Warn	40	35	45	40.00	5.00	20	20	42	27.33	12.70	21	20	30	23.75	5.45
Help	15	10	26	17.00	8.19	23	18	20	20.33	2.52	3	20	15	12.50	9.01
Fav	10	30	14	18.00	10.58	30	15	14	19.67	8.96	6	10	20	12.08	7.11
POS	65	40	55	53.33	12.58	26	32	30	29.33	3.06	34	10	22	21.92	11.88
Pref	50	26	29	35.00	13.08	23	25	30	26.00	3.61	8	5	10	7.50	2.50
SbS	15	38	38	30.33	13.28	23	30	22	25.00	4.36	3	5	10	5.83	3.82
<b>TOTAL</b>	536.0					365.7					207.8				



Table 6.6-4 Time (in min) spent by subjects in implementing the usability-related functionalities of their project

	NP (P1)	NP (P2)	NP (P3)	NP avg	NP stdv	PP (P1)	PP (P2)	PP (P3)	PP avg	PP stdv	FP (P1)	FP (P2)	FP (P3)	FP avg	FP stdv
Abort	5	180	20	68.33	97.00	30	90	15	45.00	39.69	30	45	8	27.67	18.61
Undo	40	300	90	143.33	137.96	120	90	150	120.00	30.00	71	47	32	50.03	19.93
Comm	10	140	110	86.67	68.07	60	45	53	52.67	7.51	0	57	10	22.22	30.25
Prog	30	270	90	130.00	124.90	90	60	60	70.00	17.32	30	47	15	30.56	15.84
Status	40	60	5	35.00	27.84	65	85	18	56.00	34.39	46	50	12	35.90	20.81
Warn	20	105	65	63.33	42.52	30	60	31	40.33	17.04	29	40	10	26.19	15.14
Help	5	120	25	50.00	61.44	35	50	22	35.67	14.01	6	40	10	18.57	18.68
Fav	5	34	120	53.00	59.81	20	20	60	33.33	23.09	29	10	20	19.52	9.29
POS	40	90	155	95.00	57.66	105	60	43	69.33	32.04	151	50	20	73.81	68.87
Pref	10	45	34	29.67	17.90	35	30	22	29.00	6.56	11	30	10	17.14	11.16
SbS	5	25	22	17.33	10.79	20	45	20	28.33	14.43	14	20	5	13.10	7.57
TOTAL				771.7					579.7					334.7	

Table 6.6-5 Time (in min) spent by subjects in testing the usability-related functionalities of their project

	NP (P1)	NP (P2)	NP (P3)	NP avg	NP stdv	PP (P1)	PP (P2)	PP (P3)	PP avg	PP stdv	FP (P1)	FP (P2)	FP (P3)	FP avg	FP stdv
Abort	20	40	10	23.33	15.28	10	10	5	8.33	2.89	2	1	3	2.00	1.00
Undo	60	60	45	55.00	8.66	10	15	45	23.33	18.93	5	1	5	3.67	2.31
Comm	30	45	45	40.00	8.66	10	10	45	21.67	20.21	1	1	1	1.00	0.00
Prog	10	130	45	61.67	61.71	50	5	45	33.33	24.66	1	3	1	1.67	1.15
Status	10	40	3	17.67	19.66	10	5	25	13.33	10.41	2	1	2	1.67	0.58
Warn	45	55	30	43.33	12.58	10	5	15	10.00	5.00	1	1	2	1.33	0.58
Help	10	20	8	12.67	6.43	10	5	15	10.00	5.00	1	1	2	1.33	0.58
Fav	10	45	50	35.00	21.79	10	5	30	15.00	13.23	1	1	1	1.00	0.00
POS	100	60	60	73.33	23.09	80	30	45	51.67	25.66	10	1	10	7.00	5.20
Pref	20	30	30	26.67	5.77	10	5	25	13.33	10.41	2	1	2	1.67	0.58
SbS	10	30	20	20.00	10.00	10	5	5	6.67	2.89	2	1	1	1.33	0.58
TOTAL				408.7					206.7					23.7	

### 6.6.2 Design quality data

Table 6.6-6 shows the mean scores of the designs of subjects who didn't apply the proposed process (NP), those who applied it partially (PP) and those who applied it in full (FP).

Table 6.6-6 Average quality scores (scale of 1-to-5) for resulting designs for all projects

	NP	PP	FP
Adequate Responsibility Allocation (ARA)	2.33	2.33	5.00
Diagram Readability (DR)	3.33	2.67	4.67
Notation Correctness (NC)	4.00	3.67	4.67
Over-all quality score (avg)	3.22	2.89	4.78

The data above was obtained by averaging the scores obtained by the designs of each subject (three per type of process applied) as shown in Table 6.6-7.

Table 6.6-7 Quality score (1 min, 5 max) of each project's design

	NP (P1)	NP (P2)	NP (P3)	NP avg	NP stdv	PP (P1)	PP (P2)	PP (P3)	PP avg	PP stdv	FP (P1)	FP (P2)	FP (P3)	FP avg	FP stdv
ARC	2	2	3	2.33	0.58	2	3	2	2.33	0.58	5	5	5	5.00	0.00
DR	3	3	4	3.33	0.58	4	2	2	2.67	1.15	4	5	5	4.67	0.58
NC	4	4	4	4.00	0.00	4	4	3	3.67	0.58	5	5	4	4.67	0.58
avg				3.22					2.89					4.78	
stdv				0.83					0.93					0.44	

### 6.6.3 Perceived complexity data

Table 6.6-8 shows the averages of the answers to questions 1 and 2 of the questionnaires shown in section 6.4.3, namely ranking the complexity that they found when designing and implementing the usability features. The row labeled "Design" represents the average of the 1-to-5 ranking given by subjects about how complex they found designing the usability features, while the "Implementation" row shows their answers regarding implementation of the features. The columns represent the type of process that was used, NP for no process, PP for the partial version of the process and FP for the full process.

Table 6.6-8 Average perceived complexity of all functional usability features as reported by subjects who didn't apply the proposed process (NP), who applied a partial process (PP) and who applied the full process (FP)

	Subjects who didn't apply the process (NP)	Subjects who applied the partial process (PP)	Subjects who applied the full process (FP)
Design	3.32	2.70	1.73
Implementation	3.33	2.62	1.76
avg	3.33	2.66	1.74

The data in Table 6.6-8 was obtained by averaging the subjects' scores of all functional usability features. These averages, as well as the individual scores given for each feature are shown in Table 6.6-9 and

Table 6.6-10. The first three columns in each section represent a project (P1, P2, P3), followed by the mean (avg) and standard deviation (stdv) for each type of process that was applied (NP, PP, FP). At the bottom of each table is the average across all features (shown in Table 6.6-8) along with their standard deviations.

Table 6.6-9 Complexity to implement each feature (on a 1-5 scale) as perceived by each subject

	NP					PP					FP				
	(P1)	(P2)	(P3)	avg	stdv	(P1)	(P2)	(P3)	avg	stdv	(P1)	(P2)	(P3)	avg	stdv
Abort	3	2	4	2.77	1.20	2	2	4	2.67	1.15	2	1	2	1.67	0.58
Undo	4	3	4	3.75	0.48	4	3	4	3.68	0.59	3	4	3	3.33	0.58
Comm	4	3	3	3.35	0.61	4	3	2	3.00	1.00	1	1	2	1.33	0.58
Prog	4	3	3	3.42	0.56	3	3	3	2.90	0.17	3	2	1	2.00	1.00
Status	4	2	2	2.55	1.31	3	2	2	2.23	0.40	2	4	2	2.67	1.15
Warn	3	2	3	2.43	0.74	1	3	4	2.78	1.34	1	3	1	1.67	1.15
Help	1	3	1	1.73	1.27	1	2	2	1.78	0.38	1	1	1	1.00	0.00
Fav	4	4	3	3.68	0.59	1	1	3	1.78	1.07	1	1	1	1.00	0.00
POS	5	5	5	5.13	0.23	5	4	4	4.47	0.81	3	2	2	2.33	0.58
Pref	4	5	2	3.68	1.53	3	2	3	2.57	0.51	1	1	1	1.00	0.00
SbS	4	4	4	4.02	0.03	1	2	2	1.78	0.38	1	1	1	1.00	0.00
AVG	3.32					2.70					1.73				
STDV	1.16					1.05					0.94				

Table 6.6-10 Complexity to implement each feature (on a 1-5 scale) as perceived by each subject

	NP					PP					FP				
	(P1)	(P2)	(P3)	avg	stdv	(P1)	(P2)	(P3)	avg	stdv	(P1)	(P2)	(P3)	avg	stdv
Abort	4	3	4	3.68	0.59	3	4	3	3.23	0.68	2	3	2	2.33	0.58
Undo	4	3	4	3.67	0.58	5	3	3	3.80	1.39	2	2	2	2.00	0.00
Comm	4	3	3	3.35	0.61	4	3	2	3.00	1.00	2	1	1	1.33	0.58
Prog	5	4	3	4.13	1.21	5	2	4	3.80	1.71	2	5	1	2.67	2.08
Status	4	1	2	2.35	1.55	3	1	2	1.90	0.85	1	2	3	2.00	1.00
Warn	3	2	3	2.57	0.51	3	2	3	2.57	0.51	1	3	2	2.00	1.00
Help	1	2	2	1.67	0.58	1	1	1	1.12	0.20	1	1	1	1.00	0.00
Fav	4	3	4	3.68	0.59	1	3	2	2.12	0.83	2	1	1	1.33	0.58
POS	5	5	5	5.00	0.00	3	4	3	3.23	0.68	3	2	2	2.33	0.58
Pref	3	4	2	2.90	1.01	3	2	2	2.23	0.40	1	1	1	1.00	0.00
SbS	4	3	4	3.68	0.59	1	2	2	1.78	0.38	1	1	2	1.33	0.58
AVG	3.33					2.62					1.76				
STDV	1.11					1.12					0.90				

## 6.7 Analysis

This section presents the results of comparing the averages obtained for the metrics presented in section 6.6, namely: development time, design quality and perceived complexity. It contrasts the results of using the full usability-oriented process (FP) versus not using it (NP) and using it partially (PP) for all three metrics, in order to test the hypotheses presented in section 6.2.

This analysis was performed by executing Kruskal-Wallis (KW) tests over the aforementioned data groups (NP, PP and FP) for each metric, globally and per development phase, with a set significant level of 0.05 and a confidence interval of 95%. Further (adjusted)

pairwise testing was conducted for every case in which the result of KW was significant, to determine exactly which of the groups were different from one another.

The hypotheses for this work focus on comparing the use of the FP against using NP and PP. While they are not concerned with comparing the PP to NP, the pairwise KW test provides p-values for all combinations (FP-NP, FP-PP and PP-NP), so they will all be presented and discussed in this section.

Additional testing through the Tamhane test was performed in relevant cases.

Section 6.7.1 presents the analysis of the development time data, section 6.7.2 for the design quality data, and finally, section 6.7.3 analyzes the data for perceived complexity of the functional usability features.

### 6.7.1 Development time data analysis

The next section presents an analysis for the development time data for the entire project development, followed by an analysis for the development time data over the individual project phases: analysis, design, implementation and testing.

#### 6.7.1.1 Global Analysis

Figure 6.7-1 shows the average total time taken by subjects to develop the usability related parts of their projects (analysis, design, implementation and testing phases combined).

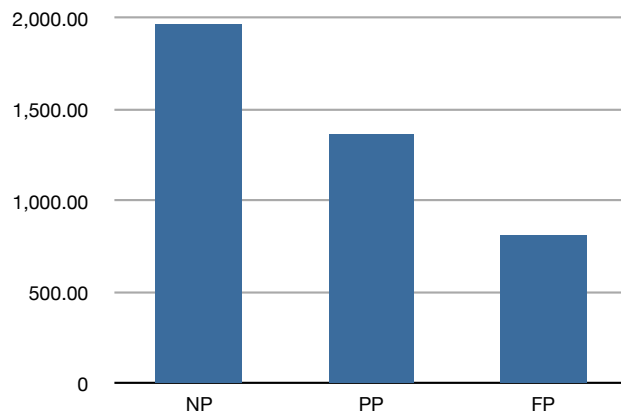
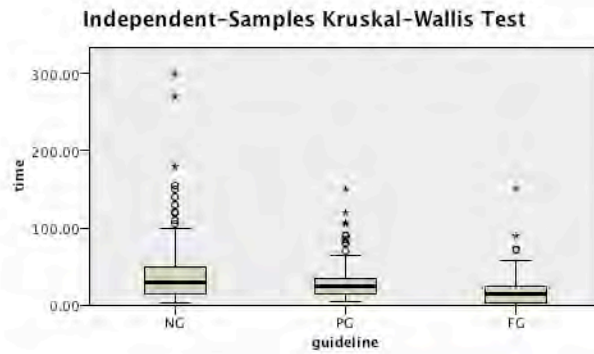


Figure 6.7-1 Average time to develop the usability of a project.

The subjects who used NP took longer in average than subjects who used a PP, who, in turn, took longer to develop their full projects than their counterparts with the FP.

A Kruskal Wallis test (applied over all 395 data points from Table 6.6-2 to Table 6.6-5, separated in the three groups, NP, PP and FP) shows that the means differ across the groups with a significance of  $p < 0.05$ . Figure 6.7-2 illustrates these results.



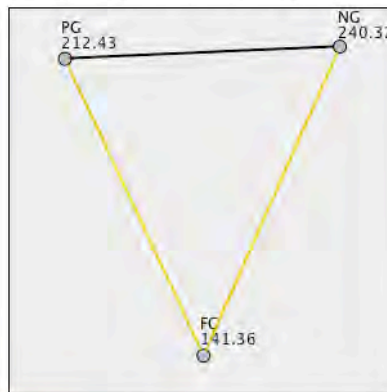
Total N	395
Test Statistic	52.888
Degrees of Freedom	2
Asymptotic Sig. (2-sided test)	.000

1. The test statistic is adjusted for ties.

Figure 6.7-2 KW test results for comparison of means across groups for total development time of usability features

Even if there is proof of difference in means, it must be determined which groups were different from each other within the three. For this purpose, the KW test was performed for pair-wise comparisons. Figure 6.7-3 illustrates the results of this test, which show that the differences in means are statistically different only for NP vs. FP and PP vs. FP, but no evidence of a statistically significant difference in means exists for the PP-NP pair.

Pairwise Comparisons of guideline



Each node shows the sample average rank of guideline.

Sample1-Sample2	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj.Sig.
FG-PG	71.071	14.057	5.056	.000	.000
FG-NG	98.958	14.030	7.053	.000	.000
PG-NG	27.887	14.057	1.984	.047	.142

Each row tests the null hypothesis that the Sample 1 and Sample 2 distributions are the same. Asymptotic significances (2-sided tests) are displayed. The significance level is .05.

Figure 6.7-3 KW test results for pairwise comparison of means for total development time of usability features

Additionally, a Tamhane test applied over this data also shows statistically significant differences between our focus pairs, FP-NP and FP-PP, as in Table 6.7-1. Furthermore, though with a lower confidence, the FP-PP pair is also confirmed different through this test.

Table 6.7-1 Tamhane test results for pairwise comparison of means for total development time of usability

(I) process	(J) process	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
NP	PP	13.54436*	4.61067	.011	2.4437	24.6450
	FP	26.17424*	4.40191	.000	15.5638	36.7847
PP	NP	-13.54436*	4.61067	.011	-24.6450	-2.4437
	FP	12.62989*	2.81651	.000	5.8588	19.4009
FP	NP	-26.17424*	4.40191	.000	-36.7847	-15.5638
	PP	-12.62989*	2.81651	.000	-19.4009	-5.8588

\*. The mean difference is significant at the 0.05 level.

### 6.7.1.2 Analysis per development phase

The next step in the analysis is to determine if such differences in terms of development time exist not only over-all, but also within each project phase, namely analysis, design, implementation and testing. Figure 6.7-4 shows the time data in Figure 6.7-1 from Table 6.6-1 discriminated by development phase.

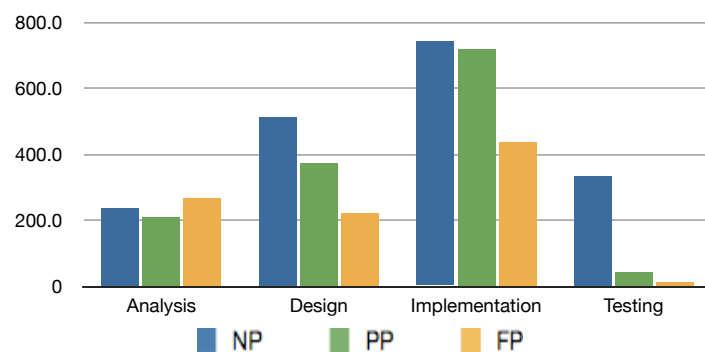


Figure 6.7-4 Average time (in min) to develop all usability-related functionality.

For all but the Analysis phase, subjects who used the FP spent less time in average than their NP and PP counterparts.

Four separate Kruskal-Wallis tests performed on this data (each for 99 data points) confirm this observation. Table 6.7-2 presents the p-values obtained when applying this test.

Table 6.7-2 p-values of Kruskal-Wallis test comparing the average time spent in each phase for all three groups

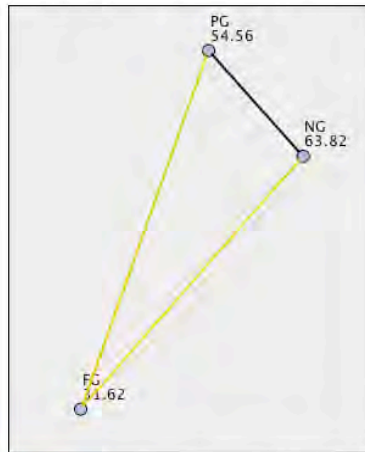
	analysis	design	implementation	testing
p-value	0.778	0.000*	0.006*	0.000*

These results show no evidence that the three means (NP, PP and FP) are statistically different for the analysis phase. For all other development phases, the test shows that the means are indeed different among the groups.

In order to determine exactly which groups are different among the three in each case, the KW test was executed for pairwise comparisons for the three development phases that proved different.

For the design phase, Figure 6.7-5 shows that the groups that are different are FP vs. PP (i.e. developers took more time to design when using the partial process than those who used the full process) and FP vs. NP (developers who didn't apply the process took longer to design than those who used the full process). Furthermore, no statistically significant difference in means was found for PP vs. NP.

**Pairwise Comparisons of guideline**



Each node shows the sample average rank of guideline.

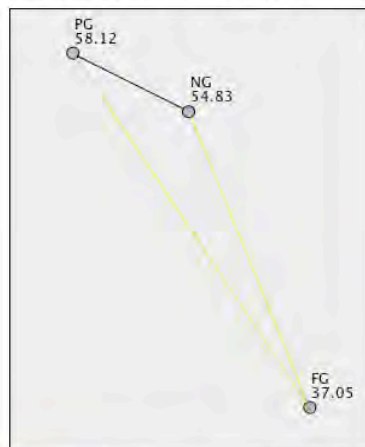
Sample1-Sample2	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj.Sig.
FG-PG	22.939	7.061	3.249	.001	.003
FG-NG	32.197	7.061	4.560	.000	.000
PG-NG	9.258	7.061	1.311	.190	.570

Each row tests the null hypothesis that the Sample 1 and Sample 2 distributions are the same. Asymptotic significances (2-sided tests) are displayed. The significance level is .05.

Figure 6.7-5 Pairwise comparisons of Kruskal Wallis test for Design phase

In the case of implementation and testing the situation was the same as shown in Figure 6.7-6 and Figure 6.7-7: the statistically significant differences in means existed between FP and PP, and also between FP and NP, but no evidence was found for the case of PP vs. NP.

**Pairwise Comparisons of guideline**



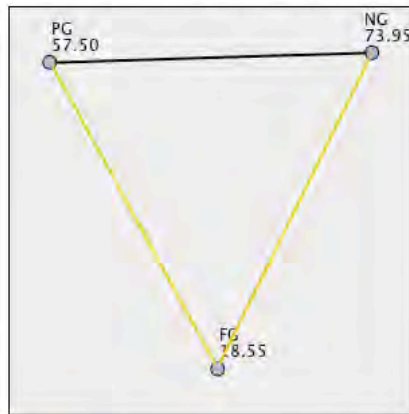
Each node shows the sample average rank of guideline.

Sample1-Sample2	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj.Sig.
FG-NG	17.788	7.062	2.519	.012	.035
FG-PG	21.076	7.062	2.984	.003	.009
NG-PG	-3.288	7.062	-.466	.642	1.000

Each row tests the null hypothesis that the Sample 1 and Sample 2 distributions are the same. Asymptotic significances (2-sided tests) are displayed. The significance level is .05.

Figure 6.7-6 Pairwise comparisons of Kruskal Wallis test for Implementation phase

**Pairwise Comparisons of guideline**



Each node shows the sample average rank of guideline.

Sample1-Sample2	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj.Sig.
FG-PG	38.955	7.009	5.558	.000	.000
FG-NG	55.409	7.009	7.906	.000	.000
PG-NG	16.455	7.009	2.348	.019	.057

Each row tests the null hypothesis that the Sample 1 and Sample 2 distributions are the same. Asymptotic significances (2-sided tests) are displayed. The significance level is .05.

Figure 6.7-7 Pairwise comparisons of Kruskal Wallis test for Testing phase

Additionally, a Tamhane test was performed over these same data and confirms the results of these four KW tests, as shown in Table 6.7-3. The Tamhane test further shows a statistically significant difference between the PP and NP pair for the testing phase, which had scored only slightly above the confidence threshold for the KW test.

Table 6.7-3 Tamhane results for pairwise comparison of means for development time of usability by phase

Analysis						
(I) process	(J) process	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
NP	PP	3.15152	2.44303	.492	-2.8505	9.1535
	FP	.24242	3.50470	1.000	-8.3840	8.8689
PP	NP	-3.15152	2.44303	.492	9.1535	2.8505
	FP	-2.90909	3.26255	.758	-10.9844	5.1662
FP	NP	-.24242	3.50470	1.000	-8.8689	8.3840
	PP	-2.90909	3.26255	.758	-5.1662	10.9844
Design						
(I) process	(J) process	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
NP	PP	15.48485	7.15736	.103	-2.2112	33.1809
	FP	29.75758*	6.60747	.000	13.2545	46.2606
PP	NP	-15.48485	7.15736	.103	-33.1809	2.2112
	FP	14.27273*	3.88458	.002	4.6838	23.8617
FP	NP	-29.75758*	6.60747	.000	-46.2606	-13.2545
	PP	-14.27273*	3.88458	.002	-23.8617	-4.6838
Implementation						
(I) process	(J) process	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
NP	PP	17.45455	14.01910	.525	-17.3383	52.2474
	FP	39.69697*	13.71485	.018	5.5609	73.8330
PP	NP	-17.45455	14.01910	.525	-52.2474	17.3383
	FP	22.24242*	7.48933	.013	3.8687	40.6161
FP	NP	-39.69697*	13.71485	.018	-73.8330	-5.5609
	PP	-22.24242*	7.48933	.013	-40.6161	-3.8687

		Testing				
(I) process	(J) process	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
NP	PP	18.36364*	5.65767	.006	4.4368	32.2905
	FP	35.00000*	4.72406	.000	23.1082	46.8918
PP	NP	-18.36364*	5.65767	.006	-32.2905	-4.4368
	FP	16.63636*	3.16344	.000	8.6805	24.5923
FP	NP	-35.00000*	4.72406	.000	-46.8918	-23.1082
	PP	-16.63636*	3.16344	.000	-24.5923	-8.6805

\*. The mean difference is significant at the 0.05 level.

The following sections break down each development phase into individual functional usability features and presents they analysis of the time data for each of them, separately.

### 6.7.1.3 Analysis phase by feature

Figure 6.7-8 presents the average time spent by subjects to perform analysis of the usability related parts of their projects, broken down by usability feature.

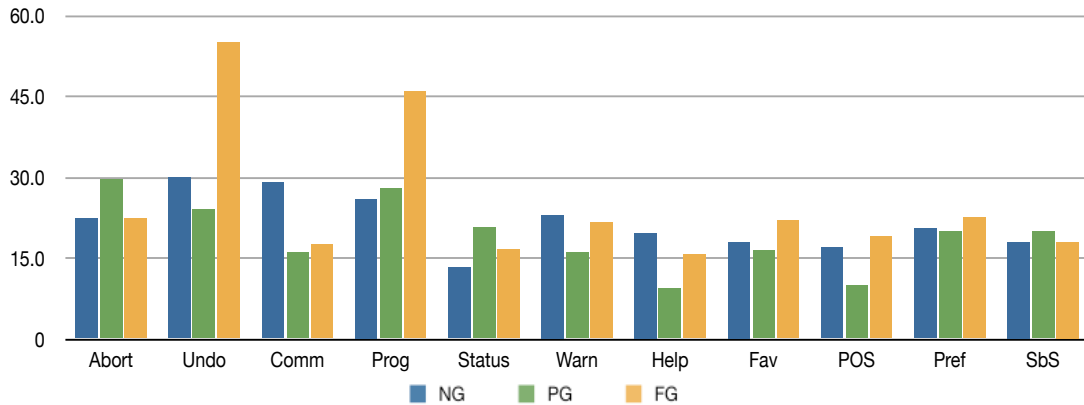


Figure 6.7-8 Average time spent on analysis, per functional usability feature

In line with the results of the KW test performed over the analysis phase as a whole shown above, there appears to be no discernible difference among the times spent in analysis for each of the types of process used, save for the Undo and Progress features. For these features, the average time spent by those using the FP appears to be greater than for those using a PP or NP, and the Commands Aggregation feature, for which the average time spent by the subject with NP appears marginally larger.

To determine if these differences are statistically significant at the desired confidence level, eleven KW tests were performed, one per usability feature, over the time averages for analysis (3 data points per average).

Table 6.7-4 shows that the resulting p-values of these tests provide no evidence of statistically significant differences among the means for any of the features.

Table 6.7-4 p-values of KW tests comparing the average time spent on each feature during analysis of projects

	Abort	Undo	Comm	Prog	Status	Warn	Help	Fav	POS	Pref	SbS
pvalue	0.579	0.967	0.421	0.837	0.073	0.739	0.107	0.505	0.864	0.904	0.380

#### 6.7.1.3.1 Design phase by feature

Figure 6.7-9 presents the average time spent by subjects to design the usability-related parts of their projects, broken down by usability feature.



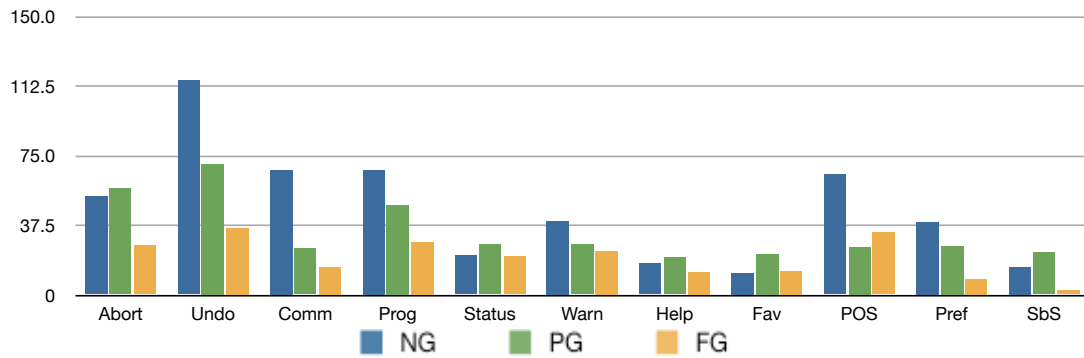


Figure 6.7-9 Average time spent on design, per functional usability feature

In line with the results of the KW test performed over the design phase as a whole, there appears to be a discernible difference among the times spent in analysis for each of the types of process used, for most of the features.

The cluster bars for Abort, Undo, Commands Aggregation, Progress, Personal Object Space, Preferences and Step by Step features in Figure 6.7-9, all show that subjects using NP or a PP spent considerably more time in average than their peers who used a FP. In the case of Warning, subjects who used a FP spent less time in average than only their NP counterparts.

In order to determine if these differences are statistically significant, KW tests were performed over the time averages (3 data points per average) for the design phase. Table 6.7-5 presents the results of these tests.

Table 6.7-5 p-values of KW test comparing the mean time spent on each functional usability feature during the design phase of the projects

	Abort	Undo	Comm	Prog	Status	Warn	Help	Fav	POS	Pref	SbS
pvalue	0.558	0.032*	0.050*	0.118	0.584	0.182	0.438	0.435	0.061† <sup>8</sup>	0.051†	0.059†

In contrast with the p-values obtained for the design phase as a whole, more than half of the p-values obtained for the mechanisms individually present no evidence of difference in means. Those that do are a subset of the group mentioned above, where the graphic suggested a decrease in time for the subjects using the FP when compared to the other two. These resulting p-values are shown in Table 6.7-5.

For the Undo, Commands Aggregation, Personal Object Space, Preferences and Step by Step features, the KW tests showed that average time spent designing it when using no process was significantly larger than when using the full process, with a confidence factor of over 95%.

For these cases which resulted in significant differences, adjusted pairwise comparisons were performed and its results shown in Table 6.7-6. In the case of Undo and Commands Aggregation, the differences were found between FP and NP. For Preferences and Step by Step, the differences were found between PP and FP.

Table 6.7-6 p-values of KW test comparing the mean time spent on each functional usability feature during the design phase of the projects

	Abort	Undo	Comm	Prog	Status	Warn	Help	Fav	POS†	Pref†	SbS†
FP-NP		0.026*	0.050*						0.076*	0.179	0.215
FP-PP		0.693	1.000						0.221	0.009*	0.014*
PP-NP		0.465	0.299						1.000	0.733	0.918

<sup>8</sup> Though the KW test for this feature scored slightly below the desired confidence value, a Tamhane test was performed to corroborate such borderline results (that still fall well under 90% confidence, also standard in this kinds of studies) and did indeed confirm pairwise differences.

### 6.7.1.3.2 Implementation phase by feature

Figure 6.7-10 shows the average time spent by subjects during implementation .

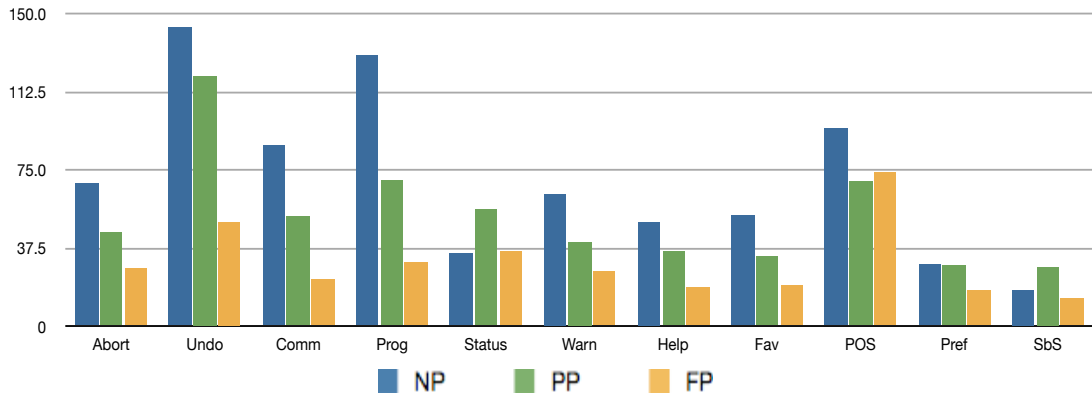


Figure 6.7-10 Average time spent on implementation, per functional usability feature

There's notable difference among the times spent on the implementation for most of the features, most prominently the Undo and Progress features, but also for Abort, Commands Aggregation, Warning, Help and Favorites.

However, after performing KW tests there is no evidence of statistically significant differences for any of the features, coming only close in the case of Undo and Progress mentioned above, but ultimately below the desired confidence level of 95%.

Table 6.7-7 p-values of KW tests comparing the mean time spent on each functional usability feature during the implementation phase of the projects

	Abort	Undo	Comm	Prog	Status	Warn	Help	Fav	POS	Pref	SbS
pvalue	0.925	0.161	0.315	0.146	0.430	0.393	0.670	0.725	0.864	0.429	0.325

### 6.7.1.3.3 Testing phase by feature

Figure 6.7-11 presents the average time spent by subjects to perform testing activities over the usability related parts of their projects, broken down by usability feature.

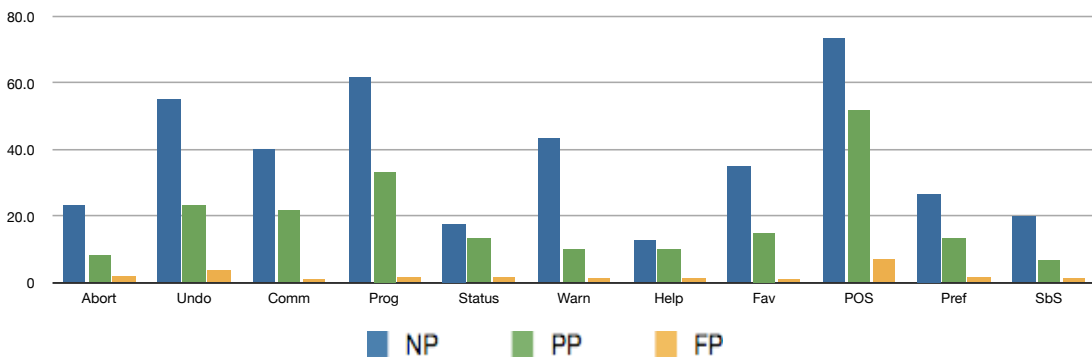


Figure 6.7-11 Average time spent on testing, per functional usability feature

In line with the results of the KW test performed over the testing phase as a whole shown at the beginning of this section, there's a discernible difference among the times spent in testing for every single functional usability feature, and it is most remarkable when comparing the NP and FP groups.

After performing the KW tests, the differences are confirmed for all features with 95% confidence, except for Progress and Status at 90%. After performing pairwise comparisons

the differences were found to be between FP and NP groups in all cases except for Status, where the adjustment for pairwise comparisons sends the p-value over 0.1 (90%).

Table 6.7-8 p-values of KW tests comparing the average time spent on each feature during the testing phase

	Abort	Undo	Comm	Prog	Status	Warn	Help	Fav	POS	Pref	SbS
pvalue	0.035*	0.030*	0.040*	0.061† <sup>9</sup>	0.064†	0.027*	0.035*	0.039*	0.048*	0.037*	0.028*

For these cases that resulted in significant differences, adjusted pair-wise comparisons were performed and its results are shown in Table 6.7-9. Additionally, values marked with † were found significant at 90% confidence

Table 6.7-9 p-values of pair-wise KW test comparing the average time spent on each feature during the testing phase

	Abort	Undo	Comm	Prog†	Status†	Warn	Help	Fav	POS	Pref	SbS
FP-NP	0.030*	0.025*	0.039*	0.088*	0.127	0.021*	0.030*	0.036*	0.048*	0.032*	0.032*
FP-PP	0.388	0.455	0.265	0.181	0.127	0.534	0.388	0.330	0.295	0.398	0.275
PP-NP	0.866	0.681	1.000	1.000	1.000	0.534	0.866	1.000	1.000	0.878	0.940

In every case which resulted in a significant value for the initial KW test, the pair-wise differences were all found between FP and NP.

## 6.7.2 Design quality data analysis

The next section presents an analysis for the design quality data for the entire project development, followed by an analysis for the development time data over the individual project phases: analysis, design, implementation and testing.

### 6.7.2.1 Global Analysis

Figure 6.7-12 shows the average of the quality marks of the resulting designs. Each bar represents the means of the three quality attributes that were proposed, namely adequate responsibility allocation, diagram readability and notation correctness for UML, by type of process applied.

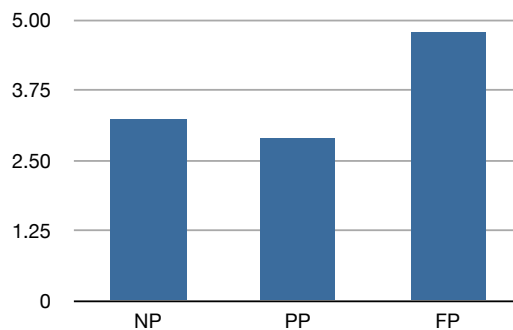
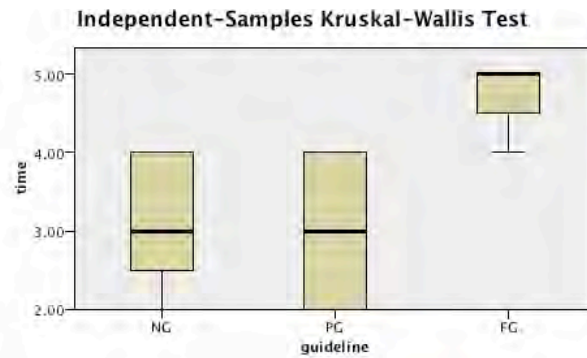


Figure 6.7-12 Average design quality marks (1-5)

The designs of subjects who used the FP scored higher than their NP and PP counterparts who scored similarly lower.

A Kruskal Wallis test was performed to determine if these three means are significantly different from one another, finding that they are, as a group, with a confidence of over 95%. Figure 6.7-13 illustrates these results.

<sup>9</sup> For the borderline cases falling below the preset 95% confidence level but still acceptable at 90%, further pairwise comparisons were conducted to determine the source of the differences.



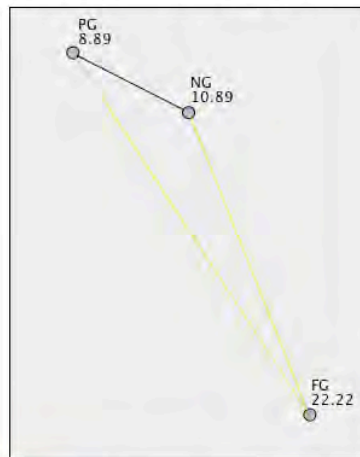
<b>Total N</b>	27
<b>Test Statistic</b>	15.893
<b>Degrees of Freedom</b>	2
<b>Asymptotic Sig. (2-sided test)</b>	.000

1. The test statistic is adjusted for ties.

Figure 6.7-13 KW test results for comparison of means across groups for over-all quality scores

After determining that there is a significant difference among the three groups, it must be determined which of the groups were different from each other within the three by performing pairwise comparisons. Figure 6.7-14 shows that there is evidence of the means being statistically different for NP vs. FP and PP vs. FP, but no evidence of difference in means exists for the PP-NP pair.

**Pairwise Comparisons of guideline**



Each node shows the sample average rank of guideline.

Sample1-Sample2	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj.Sig.
PG-NG	2.000	3.607	.554	.579	1.000
PG-FG	-13.333	3.607	-3.696	.000	.001
NG-FG	-11.333	3.607	-3.142	.002	.005

Each row tests the null hypothesis that the Sample 1 and Sample 2 distributions are the same. Asymptotic significances (2-sided tests) are displayed. The significance level is .05.

Figure 6.7-14 Pairwise KW test results for comparison of means across groups for over-all quality scores

Additionally, a Tamhane test was applied over this data and also shows statistically significant differences between our focus pairs, FP-NP and FP-PP. These results can be seen in Table 6.7-10.

Table 6.7-10 Tamhane test results for pairwise comparison of means for resulting design quality

(I) process	(J) process	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
NP	PP	.33333	.41574	.819	-.7760	1.4426
	FP	-1.55556*	.31427	.001	-2.4244	-.6867
PP	NP	-.33333	.41574	.819	-1.4426	.7760
	FP	-1.88889*	.34247	.000	-2.8450	-.9327
FP	NP	1.55556*	.31427	.001	.6867	2.4244
	PP	1.88889*	.34247	.000	.9327	2.8450

\*. The mean difference is significant at the 0.05 level.

### 6.7.2.2 Analysis per development phase

The next step in the analysis is to determine if such differences in terms of design quality exist not only over-all, but also within each project phase, namely analysis, design, implementation and testing.

Figure 6.7-15 shows the averages of the marks obtained for all three quality attributes that were proposed, namely adequate responsibility allocation, diagram readability and notation correctness for UML.

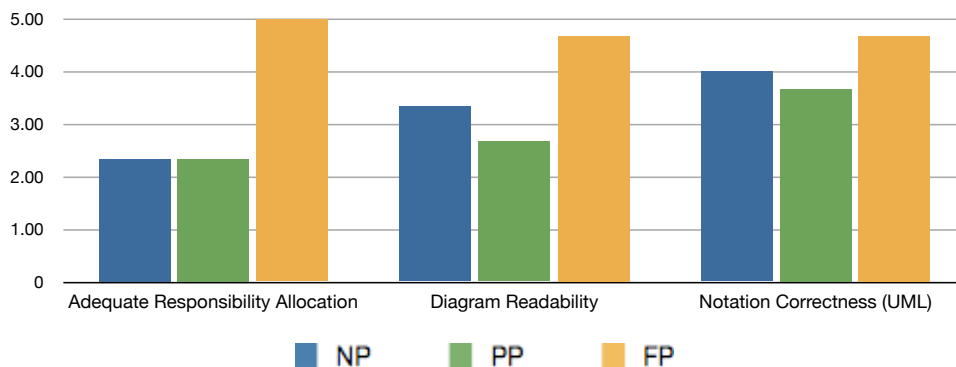


Figure 6.7-15 Averages marks obtained by the designs for the three quality attributes

The resulting designs of subjects who used the FP scored higher than their NP and PP counterparts across the board. Furthermore, the resulting designs of NP and PP subjects are similar in quality in all cases, with a slight advantage for those of NP subjects for diagram readability and notation correctness.

Three Kruskal Wallis tests were performed, one for each quality attribute, to determine the significance of these alleged difference in means for each. Table 6.7-11 shows the results.

Table 6.7-11 p-values of KW test, comparing the average time spent in each phase for all groups

	Adequate Responsibility Allocation	Diagram Readability	Notation Correctness (UML)
p-value	0.046*	0.059	0.086

The results of the test indicate that for the case of the adequate responsibility allocation attribute is there evidence to claim that the difference in means are significant among the groups. For diagram readability and notation correctness, the means cannot be proved to be different at the desired confidence levels (though they are at 90% confidence).

For the adequate responsibility allocation attribute (p value 0.046), adjusted pairwise comparisons show no difference between any of the pairs as seen in Figure 6.7-16.

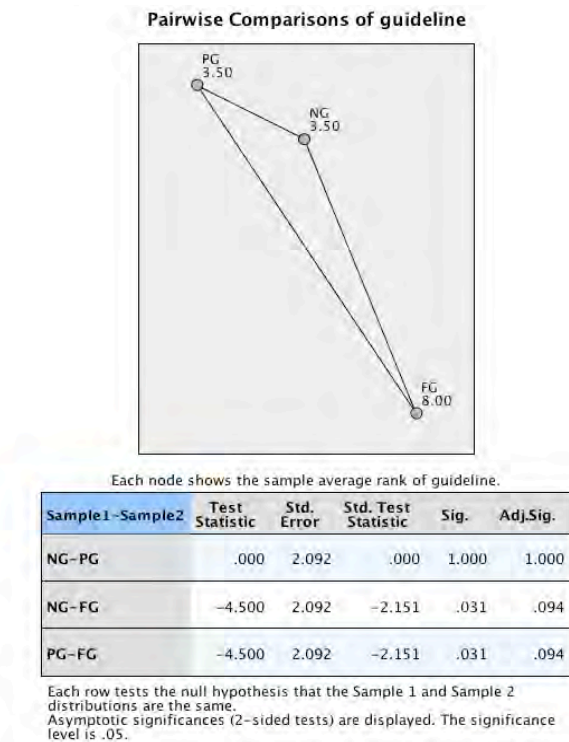


Figure 6.7-16 KW pairwise comparisons for the Adequate Responsibility Allocation attribute

Though seemingly contradictory, given that the 3-way test indicates the existence of a difference in means within the group, the adjustment factor of the pairwise KW test has, in this case, undermined the significance of the underlying pairwise differences for the given confidence interval. However, since a significant difference was found for the first test, it is of interest to find its source, albeit with an alternate test. For this purpose, a Tamhane test was performed over this same data, with the same confidence interval. Its results, shown in Table 6.7-2 indicate that the differences come from between the NP-FP and PP-FP pairs.

No evidence of difference in means was found for the NP-PP pair through this test.

Table 6.7-12 Alternate Tamhane pairwise comparisons for the Adequate Responsibility Allocation attribute

(I) process	(J) process	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
NP	PP	.00000	.47140	1.000	-1.8576	1.8576
	FP	-2.66667*	.33333	.045	-5.1941	-.1392
PP	NP	.00000	.47140	1.000	-1.8576	1.8576
	FP	-2.66667*	.33333	.045	-5.1941	-.1392
FP	NP	2.66667*	.33333	.045	1.392	5.1941
	PP	2.66667*	.33333	.045	1.392	5.1941

\*. The mean difference is significant at the 0.05 level.

### 6.7.3 Perceived complexity data analysis

The next section presents an analysis for the perceived complexity data for the entire project development, followed by an analysis for the development time data over the individual project phases: analysis, design, implementation and testing.

### 6.7.3.1 Global Analysis

Subjects were asked to rank each functional usability feature in terms of complexity during design and implementation (see section 6.4.3). Figure 6.7-17 presents the average values obtained in these questionnaires.

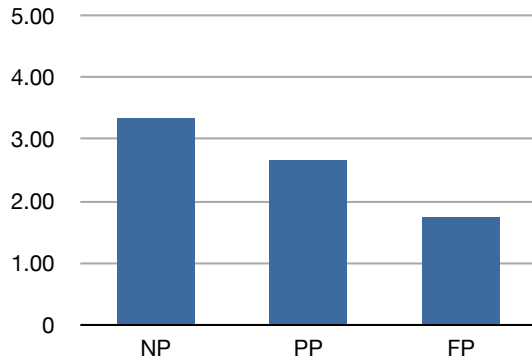
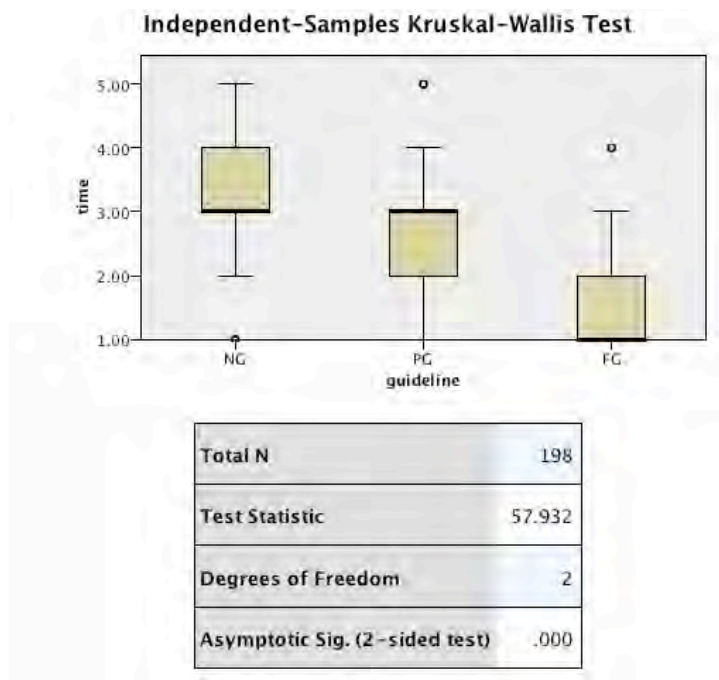


Figure 6.7-17 Average perceived complexity of all functional usability features as reported by subjects

The subjects who used NP found all features, in average, more complex to design and implement than those who used a PP, who, in turn found them more complex than their counterparts who used a FP.

A Kruskal Wallis test was performed to determine if the means differ across the three groups significantly. Figure 6.7-18 illustrates these results.

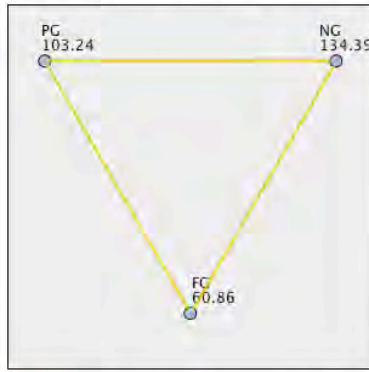


1. The test statistic is adjusted for ties.

Figure 6.7-18 KW test results for comparison of means across groups for over-all perceived complexity

Once there is proof of difference in means for the group, it must be determined which groups were different from each other. A pairwise KW test was performed, showing that the differences in means are statistically significant only for NP vs. FP and PP vs. FP, as well as for the PP-NP pair. These results are shown in Figure 6.7-19.

Pairwise Comparisons of guideline



Each node shows the sample average rank of guideline.

Sample1-Sample2	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj.Sig.
FG-PG	42.379	9.698	4.370	.000	.000
FG-NG	73.530	9.698	7.582	.000	.000
PG-NG	31.152	9.698	3.212	.001	.004

Each row tests the null hypothesis that the Sample 1 and Sample 2 distributions are the same. Asymptotic significances (2-sided tests) are displayed. The significance level is .05.

Figure 6.7-19 Pairwise comparisons of Kruskal Wallis test for over-all perceived complexity

Additionally, a Tamhane test applied over this same data also shows statistically significant differences between all three pairs, as shown in Table 6.7-3.

Table 6.7-13 Tamhane test results for pairwise comparison of means for total development

(I) process	(J) process	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
NP	PP	.69697*	.18819	.001	.2417	1.1522
	FP	1.60606*	.17525	.000	1.1821	2.0301
PP	NP	-.69697*	.18819	.001	-1.1522	-.2417
	FP	.90909*	.17698	.000	.4809	1.3373
FP	NP	-1.60606*	.17525	.000	-2.0301	-1.1821
	PP	-.90909*	.17698	.000	-1.3373	-.4809

\*. The mean difference is significant at the 0.05 level.

### 6.7.3.2 Analysis per development phase

The next step in the analysis is to determine if such differences in terms of perceived complexity exist not only over-all, but also within each project phase, namely analysis, design, implementation and testing.

Figure 6.7-20 shows the perceived complexity initially shown in Figure 6.7-17, only in this case it's discriminated by phase.

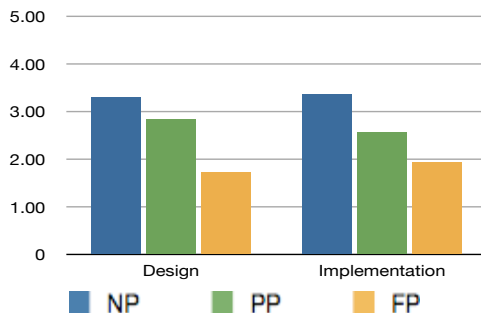


Figure 6.7-20 Average perceived complexity of all functional usability features, per phase



Users of the FP perceived the guideline as less complex than those using NP or the PP, during both the design and implementation phases.

Two KW tests was performed on these averages, one for each group, confirming that that these differences in average perception of complexity are indeed statistically different, as shown by the p-values presented in Table 6.7-14.

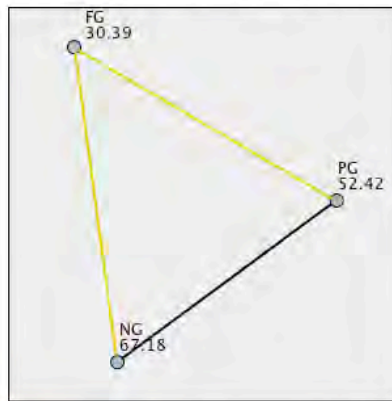
Table 6.7-14 p-values for KW test performed over the perceived confidence means of all groups

	Design	Implementation
p-value	0.000*	0.000*

To determine which groups are different among the three in both cases, the KW test was executed for pairwise comparisons.

For both the design and implementation questions, Figure 6.7-21 and Figure 6.7-22 show respectively that the groups that are different are FP vs. PP and FP vs. NP. Furthermore, no statistically significant difference in means was found for PP vs. NP.

Pairwise Comparisons of guideline



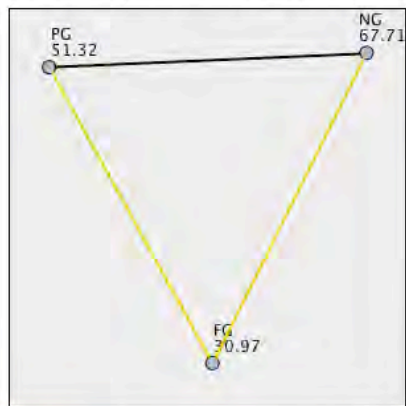
Each node shows the sample average rank of guideline.

Sample1-Sample2	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj.Sig.
FG-PG	22.030	6.876	3.204	.001	.004
FG-NG	36.788	6.876	5.350	.000	.000
PG-NG	14.758	6.876	2.146	.032	.096

Each row tests the null hypothesis that the Sample 1 and Sample 2 distributions are the same. Asymptotic significances (2-sided tests) are displayed. The significance level is .05.

Figure 6.7-21 Results of the pairwise KW test. Difference in means for perceived complexity during design.

Pairwise Comparisons of guideline



Each node shows the sample average rank of guideline.

Sample1-Sample2	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj.Sig.
FG-PG	20.348	6.872	2.961	.003	.009
FG-NG	36.742	6.872	5.347	.000	.000
PG-NG	16.394	6.872	2.386	.017	.051

Each row tests the null hypothesis that the Sample 1 and Sample 2 distributions are the same. Asymptotic significances (2-sided tests) are displayed. The significance level is .05.

Figure 6.7-22 Results of the pairwise KW test. Difference in means for perceived complexity during implementation

Additionally, a Tamhane test was performed over these same data and confirms the results of the KW test, as shown in Table 6.7-15. Furthermore, the Tamhane test further shows a statistically significant difference between the PP and NP pair, which had scored only slightly above the confidence threshold for the KW test.

Table 6.7-15 Tamhane test results for perceived complexity during design and implementation

Design						
(I) process	(J) process	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
NP	PP	.66667*	.26591	.044	.0147	1.3186
	FP	1.60606*	.24977	.000	.9934	2.2187
PP	NP	-.66667*	.26591	.044	-1.3186	-.0147
	FP	.93939*	.24977	.001	.3267	1.5521
FP	NP	-1.60606*	.24977	.000	-2.2187	-.9934
	PP	-.93939*	.24977	.001	-1.5521	-.3267
Implementation						
(I) guideline	(J) guideline	Mean Difference (I-J)	Std. Error	Sig.	95% Confidence Interval	
					Lower Bound	Upper Bound
NP	PP	.72727*	.27040	.027	.0643	1.3903
	FP	1.60606*	.24977	.000	.9934	2.2187
PP	NP	-.72727*	.27040	.027	-1.3903	-.0643
	FP	.87879*	.25455	.003	.2542	1.5034
FP	NP	-1.60606*	.24977	.000	-2.2187	-.9934
	PP	-.87879*	.25455	.003	-1.5034	-.2542

\*. The mean difference is significant at the 0.05 level.

The following two sections present the analysis of this perceived complexity data for both design and implementation, broken down by functional usability feature.

### 6.7.3.2.1 Design phase by feature

Figure 6.7-23 presents the averages of responses to the question “How would you rank this feature in terms of the complexity you encountered during the design phase?”, by type of process that was applied.

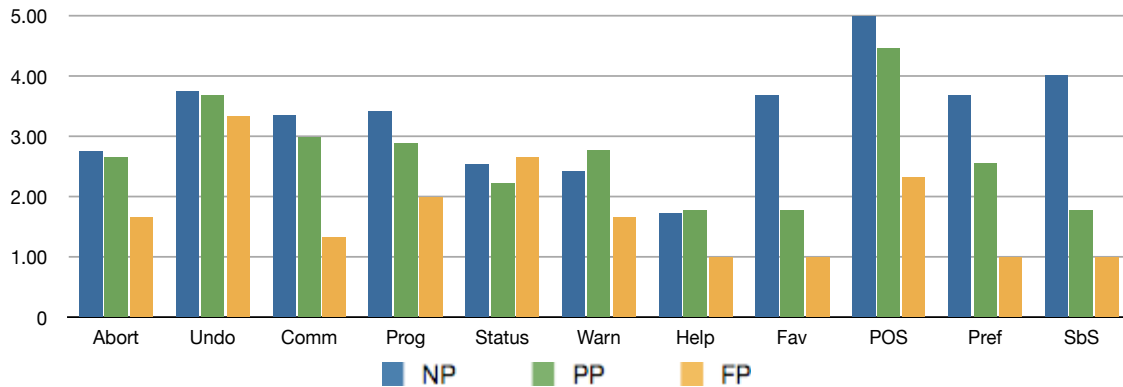


Figure 6.7-23 Average perceived complexity of each functional usability feature for the design phase

There is a significant difference in perception of the complexity to design these features between subjects using NP and those using the FP, for all but the Undo Status feature. The same holds true when comparing the average perception of those using the PP versus those using the FP. To determine if these differences are statistically significant, KW tests were performed on the means for each feature individually (3 data points per average). The results are as shown in Table 6.7-16.

Table 6.7-16 p-values for KW test over the perceived complexity means during design of each feature for all groups

	Abort	Undo	Comm	Prog	Status	Warn	Help	Fav	POS	Pref	SbS
pvalue	0.207	0.670	0.072 <sup>†10</sup>	0.113	0.953	0.464	0.543	0.127	0.032*	0.048*	0.029*

The tests provide evidence that the Personal Object Space, Preferences and Step-by-Step features were perceived as more complex to design by subjects using NP than by those using the FP. No evidence is provided by the pairwise KW test regarding difference between the PP-FP nor PP-NP groups. These results are shown in Table 6.7-17.

Table 6.7-17 p-values of KW test comparing the mean perceived complexity to design the features

	Abort	Undo	Comm	Prog	Status	Warn	Help	Fav	POS	Pref	SbS
FP-NP			0.039*						0.029*	0.053 <sup>†11</sup>	0.026*
FP-PP			0.227						0.297	0.236	1.000
PP-NP			0.957						1.000	1.000	0.283

### 6.7.3.2.2 Implementation phase by feature

Figure 6.7-23 presents the average responses to the question “How would you rank this feature in terms of the complexity you encountered during the implementation phase?”, by type of process that was applied.

<sup>10</sup> A Tamhane test was performed as a means to corroborate this borderline result still acceptable at 90% confidence (†), and again confirmed pairwise differences for this feature.

<sup>11</sup> Adjustment for the pairwise comparison sends the value below 95%, though still indicating that FP-NP is the source of the difference.

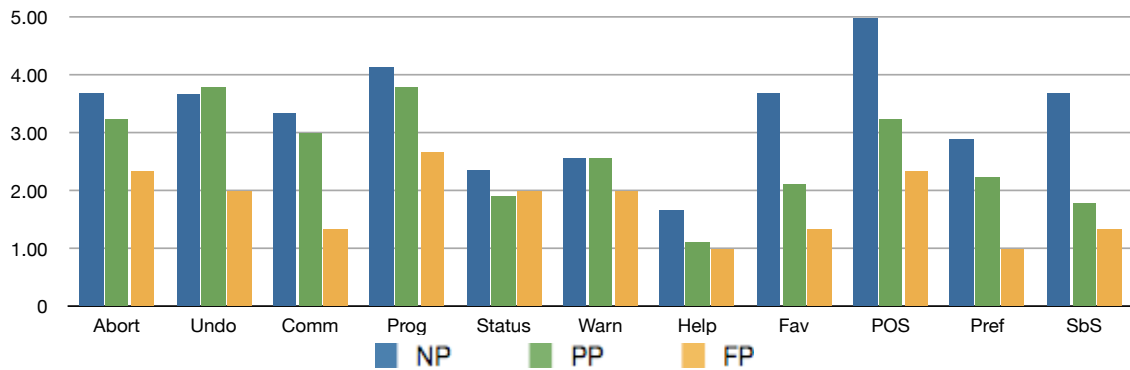


Figure 6.7-24 Average perceived complexity of each functional usability feature for the implementation phase

In line with the results of the KW test performed over the implementation phase as a whole, every functional usability feature shows a significant difference in perception of the complexity to implement when using NP versus those using the FP. Furthermore, save for the Status and Help features, the same holds true for PP vs. FP.

Individual KW tests were performed over these data to determine if these differences are statistically significant (3 data points per average) for the implementation phase. Table 6.7-18 shows the p-values obtained from performing these tests.

Table 6.7-18 KW test p-values for average perceived complexity during implementation of each feature for all groups

	Abort	Undo	Comm	Prog	Status	Warn	Help	Fav	POS	Pref	SbS
pvalue	0.048*	0.801	0.072†	0.176	0.772	0.360	0.102	0.127	0.030*	0.045*	0.032*

The differences in means are proven to be significant for the Abort, Commands Aggregation, Personal Object Space, Preferences and Step by Step features.

Further pairwise testing also confirms that all the features were perceived as more complex to implement by subjects using NP than by those using the FP. No evidence of difference in means was found between the PP and FP groups.

Table 6.7-19 pairwise KW test p-values for average perceived complexity during implementation of each feature

	Abort	Undo	Comm	Prog	Status	Warn	Help	Fav	POS	Pref	SbS
FP-NP	0.060† <sup>12</sup>		0.039*						0.025*	0.049*	0.029*
FP-PP	0.189		0.227						0.834	0.224	1.000
PP-NP	1.000		0.957						0.363	1.000	0.297

## 6.8 Results and Findings

The hypothesis for this work pertains to how using the full usability-oriented software development process helps reduce development time, improves quality of designs and reduces perceived complexity of usability features, as described in section 6.2.

In regards to time, the analysis performed on the data showed that, over all, test subjects who applied the full process developed the usability-related parts of their projects more quickly than their counterparts who didn't apply it or applied it partially, leading us to reject our first two null hypotheses, namely H0\_1 and H0\_2, respectively.

When broken down into the four project phases that were studied, it became clear that this reduction in time was coming from the design, implementation and testing phases. No discernible difference resulted from the analysis phase. This can be attributed to the fact that

<sup>12</sup> Adjustments during the pairwise comparison sends this value over the threshold yet it's still valuable for determining the source of the differences

when subjects applied the proposed process, whether partial or full, they had to invest time in understanding it and training in how to apply it, as well as the guidelines they had to use. This time was measured as part of their time invested in analysis, so, for them, the total time they spent actually analyzing would be the total *minus* this training time, resulting in a smaller analysis time than that of their counterparts who didn't apply the proposed process, and spent no time in this type of training yet took just as long in average to perform analysis on their projects. Furthermore, it could be hypothesized that when subjects who applied the proposed process develop subsequent projects, this training time is reduced and so will their total time invested in the analysis phase of their projects.

When analyzing on a feature-by-feature basis it became evident that, applying the full process saves time in all three remaining phases. Furthermore, a larger amount of this time was saved during design than during implementation, and even a more significant difference was encountered during testing.

Subjects who applied the full process created their designs more quickly than those who applied a partial version or none at all, in spite of the fact that they had to spend extra time understanding the design artifacts of the guideline before they could start designing. This leads us to believe that applying the proposed process helped them have a clearer idea about what their designs would look like more quickly than those who were unaware of it, who approached the design of their systems from scratch and had to iterate more over their designs before they were found to be implementable.

During the implementation phase the gap in time spent is less noticeable, as subjects set out to implement when they had designed in the previous phase. But when the differences are remarkable are during the testing phase, where subjects who applied the full process had a clear lead over the other two groups. This can be explained by the higher quality of the designs, which produced better code with fewer errors, and thus took less time to test. Users not applying the full process, who perhaps devised designs with unnecessary intricacy, took longer to weed out the errors than their counterparts who did apply the process in full.

Furthermore, all subjects measured any necessary re-work done to the code during testing into their testing times, further expanding the time they needed to get their applications to run error-free and also compensating for the smaller gap in the implementation phase (i.e. buggier code might take less time to implement, but it will take longer to test and fix).

In regards to quality of the resulting designs, the analysis performed on the data shows that, over-all, subjects who applied the full process produced better-quality designs than those who applied it partially and also those who did not apply it, leading us to reject our third and fourth null hypotheses respectively, H0\_2 and H0\_3.

When looking at the studied quality factors individually, however, we discovered that one of them was stronger than the other two in this regard. Subjects who applied the full process produced designs which scored significantly higher in average for adequate responsibility allocation than the other two groups. This means that the design artifacts of the guideline used in the full process, which already follow good practices in regards to object responsibility allocation, raise the over-all score of this attribute for the resulting designs of these subjects.

As for design readability and correct use of notation, even though in almost every case the designs created by subjects who applied the full process performed better in both attributes, the differences in means do not satisfy the pre-set confidence intervals of the tests that were conducted on the data.

Lastly, in regards to how complex the usability features were perceived to be by the test subjects, those who applied the full process perceived them as less complex in average than those who applied it partially and those who didn't apply it, leading us to reject our fifth and sixth null hypotheses, namely H0\_5 and H0\_6.

Breaking down the analysis into the two phases that were studied, the subjects who applied the full process found the usability features less complex both to design and implement than those who didn't. More specifically, they found them to be less complex than their partial-process counterparts, but even more so than their no-process peers. Even though no difference was demonstrated as existing between the no-process vs. partial-process pair, it is an indication that applying the process partially does help perception albeit slightly, though not to the degree achieved when applied in full.

Table 6.8-1 shows a synthesized view of the individual test result tables presented across this chapter for all tests. This table helps to categorize the guidelines according to their impact on development as explained below, judging by their p-values. A  $\sqrt{95}$  sign denotes a significant p-value at the original table with over 95% confidence, while a  $\sqrt{90}$  denotes that the confidence was 90%. In all cases the significant differences are found between the FP and NP pairs, except for design time in Preferences and Step by Step, where the differences are between the PP and NP pairs. Original tables are referenced in every row of Table 6.8-1.

When analyzing the usability features individually, Commands Aggregation, Personal Object Space, Preferences and Step by Step were found to be guidelines with the highest impact across all variables. They are represented in dark gray in Table 6.8-1. Subjects perceived these guidelines as having low complexity to design and test (hence dubbed the "easy" guidelines), while also saving significant time during both the design and testing phases.

Table 6.8-1 Summary of p-values for time and design complexity variables. " $\sqrt{}$ " represents a significant p-value

	Abort	Undo	Comm	Prog	Status	Warn	Help	Fav	POS	Pref	SbS
Time. Analysis (Table 6.7-4)											
Time. Design (Table 6.7-5)		$\sqrt{95}$	$\sqrt{95}$						$\sqrt{90}$	$\sqrt{90}$	$\sqrt{90}$
Time. Testing (Table 6.7-8)	$\sqrt{95}$	$\sqrt{95}$	$\sqrt{95}$	$\sqrt{90}$		$\sqrt{95}$	$\sqrt{90}$	$\sqrt{95}$	$\sqrt{95}$	$\sqrt{95}$	$\sqrt{95}$
Design Complexity (Table 6.7-16)			$\sqrt{95}$						$\sqrt{95}$	$\sqrt{95}$	$\sqrt{95}$
Impl. Complexity (Table 6.7-18)	$\sqrt{95}$		$\sqrt{95}$						$\sqrt{95}$	$\sqrt{95}$	$\sqrt{95}$

The Abort feature follows, shown in medium gray in Table 6.8-1. FP subjects found the Abort feature as having low complexity during the implementation phase and was also determined to save time during testing when compared to the results obtained by PP and NP subjects.

The Undo, Progress, Warning, Help and Favorites guidelines are next. Shown in light gray in Table 6.8-1, FP subjects didn't find these guidelines to have significantly low complexity to either design or implement over PP or NP, and as such were dubbed the "hard" guidelines, yet they were proven to save time during the testing phase (in the case of Undo, also during design).

Finally, the Status guideline shown in white in Table 6.8-1, appears to have had the lowest impact, as it wasn't perceived as having low complexity and also didn't save statistically significant amounts of time during development.

It's worth noting that the above categorization only considers the results of the statistical tests, even though observation alone of the figures that illustrate the source data for these tests may sometimes yield less restrictive results. For example, the Status feature, which scored below the confidence level for all tests and was categorized as having lowest impact, is shown in Figure 6.7-11 to have saved time during testing when used in full (FP), even though it did not pass the statistical tests.

## 6.9 Threats to validity

It is worth noting that while these results are highly encouraging, there is a limitation to the exactitude in which subjects were able to compartmentalize the features when measuring things like time and complexity. As the software functionalities that they truly are, usability features are woven into the software requirements specification of each project, often overlapping with other domain-specific functionality and, in some cases, with one another.

Another important factor that may be a threat to the validity of these results include the fact that the test subjects were students developing academic projects, as opposed to industry subjects working on real-life applications. The results may also not be fully generalizable due to the nature of these academic projects. Since they are mostly developed over the course of university courses, the three projects that were assigned to the students had to be of a reasonable size for them to be completed in the available time frame of the courses; around a couple of hundred function points per project. Furthermore, the problem domain for each project could potentially represent an interfering factor, yet this was mitigated by providing the students with projects that were similar yet of different domains.

Lastly, tests like Kruskal Wallis and Tamhane perform significantly better when applied over a large set of data points, but may result overly restrictive when using fewer data. Such is the case of the p-values obtained in the per-feature analyses, where each average was made up of only three data points, making the total pool of values nine points when applying the tests.

# CHAPTER 7. CONCLUSIONS AND FUTURE WORK

## 7.1 Introduction

This chapter presents the final conclusions we have reached after crafting, applying and validating our proposed approximation to a solution, followed by the lines of research it leaves open for future work.

## 7.2 Conclusions

In this doctoral thesis we propose an approximation towards solving the **open research problem of providing developers with structured, tangible guidance for including usability features with proven impact on software design into their applications.**

The inclusion of this type of usability features in to software is a complex task. The Human Computer Interaction community has proposed many recommendations, often termed Usability Patterns, detailing what needs to be present in the graphical user interface of an application for it to be usable. Useful and comprehensive as they are, these recommendations are expressed in a way that is too abstract for software engineers to actually implement

Over the past two decades, the Software Engineering community has made many attempts to bridge the gap between these HCI recommendations and their actual implementation into software. Most of these approximations have been in the form of software architectural patterns and recommendations, describing how an application should be structured internally to conform to these usability recommendations. However, software developers still face major difficulties when attempting to incorporate them into software, due in part to the fact that these recommendations are still too far removed from their actual implementation. Furthermore, the proposed solutions isolate the software design process, divorcing it entirely from many software analysis activities, when it's been shown that usability needs to be addressed throughout multiple phases of the development process.

In this work we propose an approximation to a **solution** to this problem by proposing a **Usability-oriented Software Development Process**, supported by our proposed **Software Usability Guidelines for Software Development.**



- **The Usability-oriented Software Development Process** guides software developers, throughout several analysis and design activities in incorporating specific usability features with impact on design into their software applications. It proposes activities to be carried out during the analysis and design phases of a project, regardless of the specific life cycle or development method being used. It enables software development teams to consistently and comprehensively include these specific usability features into their analysis documentation (requirements specifications, use cases, interface prototypes, story boards, etc.) and into their software design (object models, interaction diagrams, etc.).
- **The Usability Guidelines for Software Development** are used throughout the proposed process, and each of them details a possible solution for every one of the eleven usability features covered in this work. Every guideline is made up of artifacts aimed at supporting specific tasks carried out throughout the proposed process. Its most critical artifacts specify the responsibilities that the system and its parts must fulfill in order to conform to these usability features, expressed both textually and in the form of UML meta-models that are directly implementable by projects based on the MVC architecture.

The chosen usability features with impact on design are proposed by Juristo, Moreno and Sanchez-Segura in [31], where they are proven to have a considerable impact on the software logic (in terms of additional classes, methods and relationships that must be implemented) when they are included in an application. They are termed Functional Usability Features by the authors, and are grounded on solid HCI principles as shown in [32]. While there are other usability features not considered in this work that could potentially impact software functionality, our proposed approximation to a solution provides an important contribution in both the Software Engineering and HCI fields.

The main **hypothesis** for this work, which proposes that applying this proposed process with the use of the guidelines **helps reduce development time, facilitate software design and improve its quality**, was empirically validated. These validation results proved highly encouraging, enabling us to reject the corresponding null hypothesis in full.

This hypothesis was validated by having nine test subjects apply the proposed process over three university projects. Specific variables were measured for each project, including development time, design quality, and perception of complexity of the proposed guidelines.

Validation results showed that a significant amount of time was saved mostly during the design, implementation and testing phases. During analysis, the subjects had to invest additional time in understanding the proposed process and guidelines first and training in how to apply them. It could be argued, however, that when these subjects develop future projects, this training time is reduced as will their total time invested in analysis. Also, subjects who applied the proposed process created their designs more quickly as it helped them have a clearer idea about what their designs would look like more quickly than those who didn't and approached their designs from scratch, having to iterate more over them before they could implement them. Furthermore, applying the process helped developers reduce testing time remarkably as well as re-work over previous phases at the end of their projects.

In regards to quality, applying the proposed process helped developers produce designs with better responsibility allocation among objects, a key factor in software modifiability and maintainability [37].

Finally, the proposed process and guidelines helped developers perceive many of the functional usability features as not being overly complex. The perception of the complexity of

usability features is crucial as it could potentially affect a developer's disposition to embrace them within their projects.

A software tool is proposed to automate the application of the proposed usability-oriented software development process. The purpose of this tool is to make the application of the process even more efficient than when using paper versions of the Usability Guidelines for Software Development.

This work was funded by a four-year grant (FPI, BES-2007-15110) from the Spanish Ministry of Science and Innovation (MICINN), through the project titled "Tratamiento de Mecanismos de Usabilidad en las Etapas de Requisitos Y Diseño De Software" (TIN2005-00176) at the School of Computer Science of the Universidad Politecnica de Madrid.

Preliminary results from this doctoral thesis have produced three research publications in the proceedings of the following international conferences:

- 2009: ESEC/FSE Doctoral Symposium [12]
- 2010: IADIS International Conference on Interfaces and Human Computer Interaction [13]
- 2011: Jornadas de Ingenieria del Software y Bases de Datos. A Coruña [14]

### 7.3 Future Work

Future lines of research that stem from the work presented in this doctoral thesis include:

- **Contrasting automated vs. manual use of the proposed process:** The automation tool for applying the proposed usability-oriented software development process is intended to make its application, and the use of the guidelines, more efficient. It would be of great interest to compare the results of applying the process manually, as it has been done for this work, with its automated application. Specifically, the variable of development time, as measured in this work, could be observed to determine if indeed the automated tool helps developers perform their tasks faster.
- **Validating in industry projects:** The next desirable step in the validation of the approximation to the solution proposed in this work would be to test it in an industry setting. The Usability-oriented Software Development Process could be applied in several large-size real-life projects to test its impact in such a setting.
- **Incorporating functional usability features specific to mobile devices:** With the recent advances in mobile technologies, the once all-dominating personal computer now lives side-by-side with devices such as smartphones and tablet computers. These newer devices compete eagerly to helping users perform tasks that they once performed only on traditional computers. As these newer devices have vastly different user interaction paradigms, it would be of interest to asses the applicability of the proposed Usability Guidelines for Software Development for the chosen usability features under these newer conditions. For example the guideline Multi-level Help feature proposed in this work, which for certain cases provides textual help information when the user "hovers" the pointer over a specific interface item, would need to be redesigned for the fully tactile devices where "hovering" is not an option
- **Providing implementations of the designs in multiple programming languages:** Implementation artifacts such as software plug-ins are not provided as part of the proposed Usability Guidelines for Software Development. Doing so would have resulted in a solution approximation whose end result as of very limited applicability, as it would have been relevant only by developers who used the same language in which the plug-ins would have been written. However, it could be argued that if produced for enough of

the widely used long-standing languages dominating the software development sector today, they could prove to be an advantageous addition to our proposed guidelines.

- **Providing a design solution for other architectures:** The design portions of our approximation to the solution are based on an MVC architecture. While MVC is a popular architecture, and MVC-based designs are of use to development teams using this architecture but also others to which MVC designs can be translated (i.e. PAC), further work could be conducted in transforming the proposed designs for other architectures, such as SOA, three-tier, distributed, etc.
- **Estimating the effort required to include each Functional Usability Feature:** Including each of the Functional Usability Features discussed in this work by applying the proposed process represents a quantifiable effort on the part of the development team. It would be of great interest to this line of research to measure this effort for each one of the features through experimentation. This would provide developers not only with a solution approximation to help them include the features into their software, but also with an estimation of the actual cost that such an inclusion would represent, as an effort measurement can be ultimately translated into economic costs.

## CHAPTER 8. REFERENCES

- [1] Alexander, C., Ishikawa, S., and Silverstein, 1997 M. A Pattern Language, Oxford University Press, New York.
- [2] Ambler, S. 2005. The elements of UML 2.0 style. Cambridge University Press.
- [3] Bass, L. and John, B. E., 2002. Linking Usability to Software Architecture Patterns Through General Scenarios. *Journal of Systems and Software*, Volume 66, Issue 3, pp 188-197.
- [4] Bass, L. and John, B. E., Kates, J. Achieving Usability Through Software Architecture. Technical Report. CMU/SEI-2001-TR-005, March 2001.
- [5] Bass, L. et al. Unravelling the Myths of Developing Usable Software. Unpublished work.
- [6] John, B., Bass, L., Golden, E., Stoll, P. 2009. A Responsibility-Based Pattern Language for Usability-Supporting Architectural Patterns. Proceedings of the 1st ACM SIGCHI symposium on Engineering interactive computing systems
- [7] Battey, J. 1999. IBM's redesign results in a kinder, simpler web site. Retrieved from Infoworld:  
<http://www.infoworld.com/cgi-bin/displayStat.pl?pageone/opinions/hotsites/hotextr990419.htm>
- [8] Benson, C., Elman, A., Nickell, S. and Robertson, C. 2007. GNOME Human, "Interface Guidelines,"  
<http://developer.gnome.org/projects/gup/hig/1.0/index.html>
- [9] Bevan, N. (2008). UX , Usability and ISO Standards. CHI 2008 Workshop on User Experience Evaluation Methods in Product Development, 1-5. Retrieved from [http://www.cs.tut.fi/ihte/CHI08\\_workshop/slides/Bevan\\_UXEM\\_slides.pdf](http://www.cs.tut.fi/ihte/CHI08_workshop/slides/Bevan_UXEM_slides.pdf)
- [10] Bias, R. M. 2005. Cost-Justifying Usability. An Update for the Internet Age. Elsevier.
- [11] Black, J. 2002. Usability is next to profitability. Retrieved 04 10, 2009. BusinessWeek Online:  
[http://www.businessweek.com/technology/content/dec2002/tc2002124\\_2181.htm](http://www.businessweek.com/technology/content/dec2002/tc2002124_2181.htm)
- [12] Brighton, 1998. Usability Pattern Collection. <http://www.cmis.brighton.ac.uk/research/patterns/home.html>
- [13] Carvajal, L. 2009. Usability-enabling guidelines: a design pattern and software plug-in solution. Proceedings of the doctoral symposium for ESEC/FSE on Doctoral symposium. ACM, New York, NY, USA, 9-12.
- [14] Carvajal, L., Moreno, A. 2010. USABILITY-ENABLING SOFTWARE DESIGN GUIDELINES. Proceedings of the IADIS International Conference on Interfaces and Human Computer Interaction. Freiburg, Germany 26-30 July 2010.
- [15] Carvajal, L., Moreno, A. 2011. Software Design Guidelines for Usability. Jornadas de Ingenieria del Software y Bases de Datos. A Coruña, Spain. 5-7 September 2011
- [16] Cochrane Collaboration. 2003. Cochrane Reviewers' Handbook. Version 4.2.1.
- [17] Constantine, L. and Lockwood, L., 1999. Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design. Addison-Wesley. New York, USA.
- [18] Coram, T. and Lee, L. 1996 "Experiences: A Pattern Language for User Interface Design,"  
<http://www.maplefish.com/todd/papers/experiences/Experiences.html>.
- [19] Donahue, G. 2001. Usability and the bottom line. IEEE Software, 16 1, 31-37.

## CHAPTER 9. APPENDIXES

### 9.1 Example application of the proposed Usability-Oriented Software Development Process

This appendix shows how the proposed would be applied over an example project. This project is a modified version of the Home Automation System shown in Appendix 9.4.

Section 9.1.1 describes the process for this example over the Requirements elicitation and analysis activities for usability, and section 9.2 describes it for the OO software design activities for usability.

#### 9.1.1 Requirements elicitation and analysis for usability

During elicitation of this example project, the analyst(s) gather a list of functional requirements. The following requirement is among them:

Req(3) The system must control the window blinds. Window blind panels can rotate from 15 to 165 degrees. The position with greater sunlight is 90 degrees. The execution of this process normally lasts approximately 1/10th of a second per degree of rotation

The next three sections describe the results of applying the three tasks in this sub-process

#### 9.1.2 Functional usability requirements elicitation

After performing this first task, the SRS is enhanced to include usability. In the case of the example requirement shown in the previous section, it now looks as follows:

Req(3) The system must control the window blinds. Window blind panels can rotate from 15 to 165 degrees. The position with greater sunlight is 90 degrees. The execution of this process normally lasts approximately 1/10th of a second per degree of rotation, during which the system must show a progress bar indicating and continuously updating the percentage of execution.

- [20] Ferre, X., et al, 2003. A software architectural view of usability patterns. *Proceedings of INTERACT 2003*. Zurich, Switzerland. n.p.
- [21] Folmer, E. van Gorp, J. and Bosch, J., 2005. Software Architecture Analysis of Usability. *Lecture Notes in Computer Science*, Volume 3425, pp 38-58.
- [22] Fowler, M. 1999. *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)*. Addison-Wesley Professional.
- [23] Freeman, E., Bates, B. and Sierra, K. 2004. *Head-first design patterns*. O'Reilly & Associates, Inc.
- [24] Griffith, J. 2002. Online transactions rise after bank redesigns for usability. Retrieved from *The Business Journal*: <http://www.bizjournals.com/twincities/stories/2002/12/09/focus3.html>
- [25] Heckel, P., 1991. *The Elements of Friendly Software Design*, second ed. Sybex Inc, CA.
- [26] Hix, D., Hartson, H.R., 1993. *Developing User Interfaces: Ensuring Usability Through Product and Process*. John. Wiley & Sons, New York.
- [27] IEEE. 1998. Standard for a Software Quality Metrics Methodology. IEEE Std 1061.
- [28] ISO. 2000. Part 1: Quality Model. ISO 9126 Software Engineering. Product Quality.
- [29] ISO. 1998. Part 11: Guidance on Usability. ISO 9241 Ergonomic requirements for office work with visual display terminals.
- [30] ISO/IEC 25062:2006 "Common Industry Format (CIF) for usability test reports
- [31] John, B., Bass, L. and Sanchez-Segura, M., 2005. Bringing Usability Concerns to the Design of Software Architecture. *Lecture Notes in Computer Science*, Volume 3425, pp 1-19.
- [32] Juristo, N., Moreno, A. and Sanchez-Segura, M., 2007. Analysing the impact of usability on software design. *Journal of Systems and Software*, Volume 80, Issue 9, pp 1507-1516.
- [33] Juristo, N., Moreno, A. and Sanchez-Segura, M.I., 2007. Guidelines for eliciting usability functionalities. *IEEE Transactions on Software Engineering*, Volume 33, Issue 11, pp 744-758.
- [34] Juristo, N., et al., 2007. Glass Box Design: Making the Impact of Usability on Software Development, *Proceedings of INTERACT 2007*, Rio de Janeiro, Brazil. n.p.
- [35] Kitchenham, B. 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. EBSE Technical Report EBSE-2007-01. Department of Computer Science University of Durham Durham, UK
- [36] Kumar, S. et. al. / *International Journal of Engineering Science and Technology* Vol. 2(9), 2010, 4723-4729
- [37] Laasko, S.A., 2003. User Interface Designing Patterns. <[http://www.cs.helsinki.fi/u/salaakso/patterns/index\\_tree.html](http://www.cs.helsinki.fi/u/salaakso/patterns/index_tree.html)> Visited October 2004.
- [38] Larman, C. 2004 *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Edition)*. Prentice Hall.
- [39] Nielsen, J., 1993. *Usability engineering*. Morgan Kaufmann Publishers, Boston, USA.
- [40] Seffah, A. et al, 2008. Reconciling usability and interactive system architecture using patterns. *Journal of Systems and Software*, Volume 81, Issue 11, pp 1845-1852.
- [41] Seffah, A. M., 2004. The obstacles and myths of usability and software engineering. *Communications of the ACM*, Volume 47, Issue 12, pp 71-76.
- [42] Seffah, A. M., 2006. Usability measurement and metrics: A consolidated model Ahmed Seffah, Mohammad Donyae, Rex B. Kline and Harkirat K. Padda
- [43] Tidwell, J., 2005. *Designing Interfaces. Patterns for Effective Interaction*
- [44] *Design*. O'Reilly, USA.
- [45] Trenner, L. 1998. *The Politics of Usability*. London,UK:Springer
- [46] Shneiderman, B., 1998. *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, third ed. Addison Wesley, Menlo Park, CA.
- [47] SIGCHI Curricula for Human-Computer Interaction. ACM. [http://old.sigchi.org/cdg/cdg2.html#2\\_1](http://old.sigchi.org/cdg/cdg2.html#2_1)
- [48] Stevens, S. S. (19-16). On the theory of scales of measurement. *Science*. 103, 67 t-680.
- [49] Stoll, P., et al., 2009. Supporting Usability in Product Line Architectures. *Proceedings of the 13th International Software Product Line Conference*. San Francisco, CA, August 24-28, 2009
- [50] van Welie, M., 2008. *The Amsterdam Collection of Patterns in User Interface Design*. [Online] Available at: <http://www.welie.com>.
- [51] Business Process Management Initiative. Object Management Group. Business Process Model Notation. <http://www.bpmn.org/>

### 9.1.3 Usability use case modeling

If modeling use cases, the system's use case model, which may look like Figure 9.1-1 when not including usability,

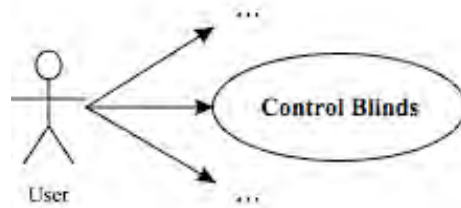


Figure 9.1-1 Example use case model not including usability

is enhanced with usability by applying the relevant parts of the Usability use case meta models for the Progress functional usability feature, shown in Figure 9.1-2.

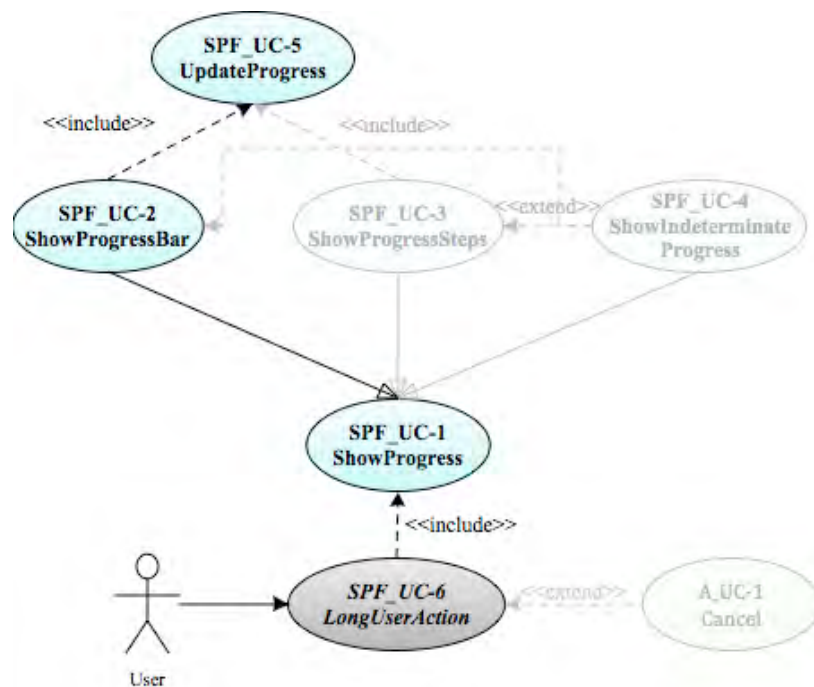


Figure 9.1-2 Usability use case meta-models for the Progress functional usability feature

and ends up looking like Figure 9.1-3, enhanced to include the required usability.

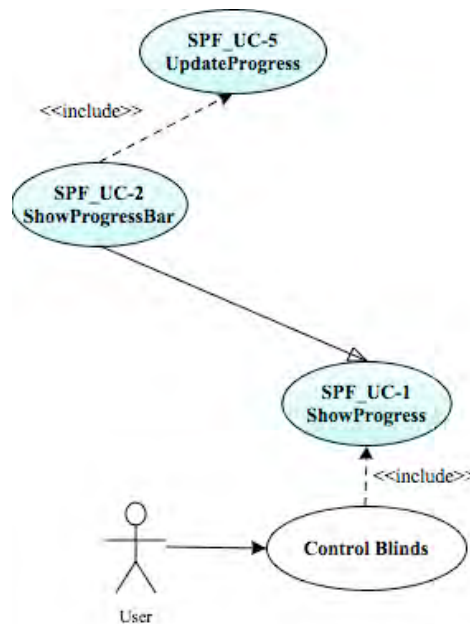


Figure 9.1-3 Example use case model including usability

### 9.1.4 Identification of system responsibilities

In this example, where the only usability requirement is to show a progress bar when operating the window blinds, the system analysts would only consider the relevant System Responsibilities for the Progress feature, ignoring the rest, shown in gray in Table 9.1-1,

Table 9.1-1 System Responsibilities for Progress functional usability feature

System Responsibilities List for Progress Feedback	
Determine which tasks will require progress	
The system must know which system actions might take long to execute	
Calculate and provide progress information	
The system must provide progress information for each action by using all available information	
Provide cancel option	
The system must allow users to cancel on-going actions	
Provide textual information	
The system must provide information about the task during progress display	
Provide indeterminate progress information	
The system must provide indeterminate progress info for tasks requiring it or when no other alternative is available	

## 9.2 OO software design activities for usability

Once the System Responsibilities are determined, the process continues with the last three tasks, as described in the next three sections

### 9.2.1 Identification of High-level design component responsibilities for usability

If producing this intermediate output, the software designers consider only the relevant High-level design component responsibilities for usability, based on the System Responsibilities that were considered in the previous task, ignoring the rest, shown in gray in Table 9.2-1.

Table 9.2-1 High-level design component responsibilities for Progress functional usability feature

System Responsibility	High-level design component responsibilities for usability
Determine which tasks will require progress inf.	The <i>UI Component</i> is responsible for knowing (from a pre-established list) whether an invoked action is among those that could potentially be 'long' (>2s)
Calculate and provide progress information	The <i>UI Component</i> is responsible for listening for calls to invoke these long actions and for ordering their... If the action is among the potentially 'long', the <i>UI</i> must call unto an alternate <i>Monitoring Component</i> ...
Provide cancel option	The component responsible for displaying the progress (be it the <i>UI</i> or an alternate <i>Progress Component</i> ) must provide a cancel option for the actions it knows to require one
Provide textual info	The component responsible for displaying progress must also know of and display any needed textual info ...
Provide indeterminate progress information	When the <i>UI</i> component (or alternate <i>Progress Component</i> ) first displays the progress, it must do ...



## 9.2.2 Identification of Low-level design component responsibilities for usability

If producing this intermediate output, the software designers consider only the relevant Low-level design component responsibilities for usability, based on the System Responsibilities that were considered in the previous task, ignoring the rest, shown in gray in Table 9.2-2.

Table 9.2-2 Low-level design component responsibilities for Progress functional usability feature

System Responsibility	Objects					Fig
	View	ProgressIndicator	Monitor	Controller	DomainClass	
Determine which tasks will require progress information	1. The View must listen for invocation of actions and must determine (from a preexisting list) if the action being called could be potentially long.					
Calculate and provide progress information	1. Notify the Controller when a long action has been invoked. 2. Ask the Monitor class to wait a specified amount of time (2s) ...	6. The ProgressIndicator subscribes to the corresponding DomainClass for ...	3a. The Monitor class starts up a clock and notifies the view after the time (2s) has....	3b. The Controller invokes the action, calling the ...	4. The DomainClass starts executing the invoked action ...	3, 4
Provide cancel option	1. When the View creates the ProgressIndicator it ...	2. ProgressIndicator will enable a 'cancel' button.				3, 4
Provide textual information	1. When the View creates the ProgressIndicator it must ...	2. The ProgressIndicator holds this text ...				3, 4
Provide indeterminate progress info	1. Whenever a ProgressIndicator (that is not undetermined) is created ...					3, 4

## 9.2.3 Object Oriented software design for usability

The OO class diagram being produced, which may look like Figure 9.2-1 when not yet including usability,

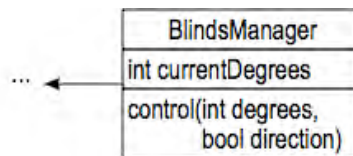


Figure 9.2-1 Example use case model not including usability

is enhanced with usability by applying the relevant parts of the OO software design meta models for usability for the Progress feature, whose class diagram is shown in Figure 9.2-2.

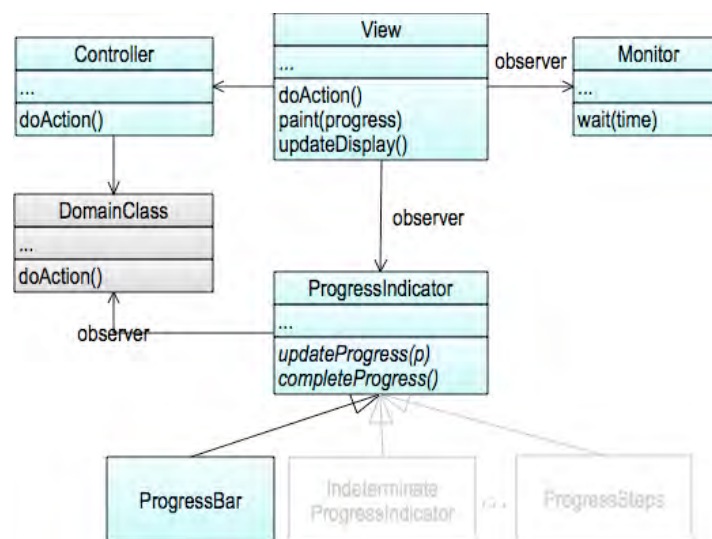


Figure 9.2-2 Usability OO software design meta models for usability. Class Diagram.

and ends up looking like Figure 9.2-3, enhanced to include the required usability.

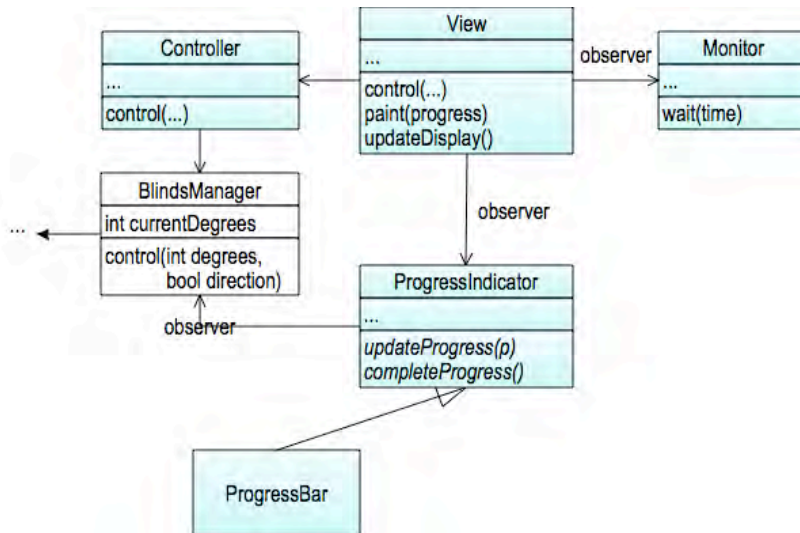


Figure 9.2-3 Example class diagram including usability

Similarly, in the case of the sequence diagram, when applying the relevant parts of the OO software design meta models for usability for the Progress feature, whose sequence diagram is shown in Figure 9.2-4.

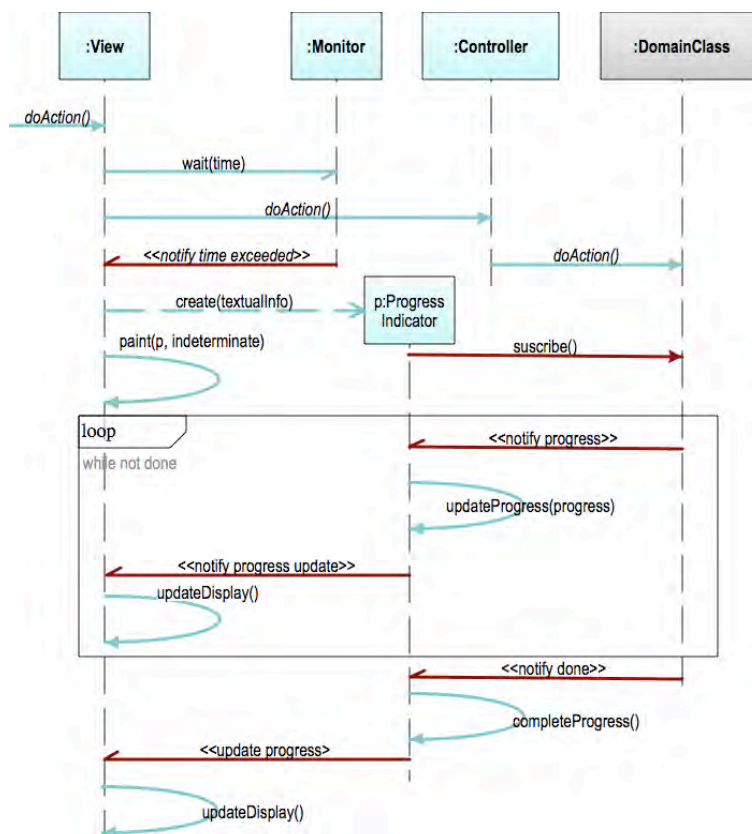


Figure 9.2-4 Usability OO software design meta models for usability. Class Diagram.

ends up looking like Figure 9.2-5, enhanced to include the required usability.

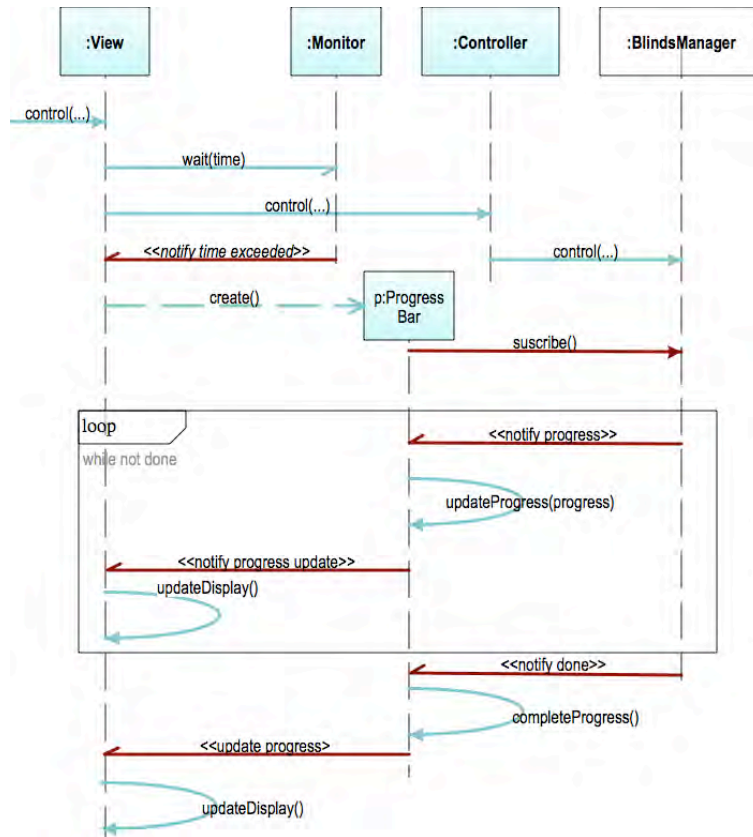


Figure 9.2-5 Example sequence diagram including usability

### 9.3 Full results of systematic literature review

This appendix presents the full results of the systematic literature review for this doctoral thesis, for all the proposed key phrases that turned up relevant (and non repeated) results: Software architecture usability, Usability patterns software design and Architectural patterns usability. Initially selected studies appear highlighted. Pages from which no results were selected (or which only contained repeated results from previous key phrases) are omitted. The key phrase for each Google scholar search result page appears on the top-left corner.

Web Images Videos Maps Books Translate Gmail more ▼ laura.carvajal@gmail.com | Scholar Preferences | My Account | Sign out

Google scholar software architecture usability Search Advanced Scholar Search

Scholar Articles excluding patents anytime include citations Create email alert Results 1 - 100 c

**Linking usability to software architecture patterns through general scenarios** [PDF] from psu.edu

L Bass... - Journal of Systems and Software, 2003 - Elsevier  
**Usability** is an important quality attribute to be considered during **software architecture** design. Up to this point, **usability** has been served only by separating a system's user interface from its functionality to support iterative design. However, this has the effect of pushing ...  
 Cited by 92 - Related articles - All 10 versions - Import into RefWorks

**Scenario-based assessment of software architecture usability** [PDF] from psu.edu

E Folmer, J Van Gorp... - Bridging the Gaps Between Software ..., 2003 - Citeseer  
 Scenario-based Assessment of **Software Architecture Usability** Eelke Folmer, Jilles van Gorp, Jan Bosch Department of Mathematics and Computing Science University of Groningen, PO Box 800, 9700 AV the Netherlands mail@eelke.com, Jilles@cs.rug.nl, ...  
 Cited by 22 - Related articles - View as HTML - All 18 versions - Import into RefWorks

**Pattern-oriented software architecture: a system of patterns** [PDF] from ispras.ru

F Buschmann, R Meunier, H Rohnert, P Sommerlad... - 2008 - Wiley-India  
 Cited by 4090 - Related articles - All 13 versions - Import into RefWorks

**Achieving usability through software architecture** [PDF] from psu.edu

... - MELLON UNIV PITTSBURGH PA SOFTWARE ... - 2001 - Citeseer  
 ... The goal of this work is to achieve better system **usability** through design decisions embodied in the **software architecture** ... Hence, understanding the relationship between **software architecture** and **usability** is important to ensure that the system ultimately achieves it ...  
 Cited by 81 - Related articles - View as HTML - BL Direct - All 25 versions - Import into RefWorks

**Supporting usability through software architecture**

L Bass... - Computer, 2001 - ieeexplore.ieee.org  
 ARCHITECTURAL PATTERNS An architectural pattern expresses some fundamental relationships among **software** elements. It provides a set of required components, specifics of their relationships, and the responsibilities necessary to implement the relationships. **Software** ...  
 Cited by 36 - Related articles - BL Direct - All 8 versions - Import into RefWorks

**A framework for classifying and comparing software architecture evaluation methods** [PDF] from psu.edu

MA Babar, L Zhu... - Software Engineering Conference ..., 2004 - ieeexplore.ieee.org  
 ... maintainability, reliability, **usability**, performance, flexibility etc.) of large **software** systems are largely constrained by the systems' SA [3]. Since SA plays a significant role in achieving system wide quality attributes, it is very important to evaluate a system's **architecture** with regard ...  
 Cited by 99 - Related articles - All 12 versions - Import into RefWorks

**A framework for capturing the relationship between usability and software architecture** [PDF] from psu.edu

E Folmer, J Van Gorp... - Software Process: ..., 2003 - Wiley Online Library  
**Usability** is increasingly recognized as an essential factor that determines the success of **software** systems. Practice shows that for current **software** systems, most **usability** issues are detected during testing and deployment. Fixing **usability** issues during this late stage of the ...  
 Cited by 39 - Related articles - BL Direct - All 12 versions - Import into RefWorks

**Software architecture analysis of usability** [PDF] from psu.edu

E Folmer, J van Gorp... - Engineering Human Computer ..., 2005 - Springer  
 Abstract. Studies of **software** engineering projects show that a large number of **usability** related change requests are made after its deployment. Fixing **usability** problems during the later stages of development often proves to be costly, since many of the necessary changes require ...  
 Cited by 34 - Related articles - BL Direct - All 27 versions - Import into RefWorks

**The value of a usability-supporting architectural pattern in software architecture design: a controlled experiment** [PDF] from psu.edu

E Golden, BE John... - international conference on Software ..., 2005 - portal.acm.org  
 ... Keywords Controlled experiment, **usability**, **software architecture**, design pattern, modification, ... [9] John, BE and L. Bass, "Avoiding 'We can't change THAT!': **Software Architecture** and **Usability**", tutorial materials presented at CHI 2003, Ft. Lauderdale, FL, April 5-10, 2003. ...  
 Cited by 25 - Related articles - BL Direct - All 11 versions - Import into RefWorks

**Scenario-based analysis of software architecture** [PDF] from psu.edu

R Kazman, G Abowd, L Bass... - Software, IEEE, 1996 - ieeexplore.ieee.org  
 ... portability. To address this problem, we have developed the **Software Architecture** Analysis Method, an approach that uses scenarios to gain information about a system's ability to meet desired quality attributes. Scenarios-brief ...  
 Cited by 391 - Related articles - BL Direct - All 32 versions - Import into RefWorks

**Bringing usability concerns to the design of software architecture** [PDF] from psu.edu

BE John, L Bass, MI Sanchez-Segura... - Computer Interaction and ..., 2005 - Springer  
 R. Bastide, P. Palanque, and J. Roth (Eds.): EHCI-DSVIS 2004, LNCS 3425, pp. 1-19, 2005.  
 © IFIP International Federation for Information Processing 2005 ... **Bringing Usability** Concerns to the Design of **Software** ... Bonnie E. John1, Len Bass2, Maria-Isabel ...  
 Cited by 28 - Related articles - BL Direct - All 13 versions - Import into RefWorks

**ArchJava: connecting software architecture to implementation** [PDF] from psu.edu

J Aldrich, C Chambers... - Software Engineering, 2002 - ieeexplore.ieee.org  
 Page 1, Arch Java: Connecting **Software Architecture** to Implementation Jonathan Aldrich Craig Chambers David Notkin ... Abstract **Software architecture** describes the structure of a system, enabling more effective design, program understanding, and formal analysis. ...  
 Cited by 406 - Related articles - BL Direct - All 31 versions - Import into RefWorks

**A survey on software architecture analysis methods** [PDF] from psu.edu

L Dobrica... - IEEE Transactions on software Engineering, 2002 - computer.org  
 ... According to this, there are six categories of characteristics (functionality, reliability, **usability**, efficiency, maintainability, and portability), which are divided into subcharacteristics. ... 2.2 **Software Architecture** Definition and Description. Definition. ...  
 Cited by 289 - Related articles - BL Direct - All 22 versions - Import into RefWorks

**Usability and software architecture**

BE John... - Behaviour and Information Technology, 2001 - ingentaconnect.com  
 The role of **software architecture** with respect to **usability** has evolved over the past 20 years. The architectures of the 1980s and early 1990s assumed that **usability** was primarily a property of the presentation of information. Therefore, simply separating the presentation from the ...  
 Cited by 18 - Related articles - BL Direct - All 6 versions - Import into RefWorks

**Software architecture for hard real-time applications: cyclic executives vs. fixed priority executives** [PDF] from douglock

CD Locke - Real-Time Systems, 1992 - Springer  
 ... Manufactured in The Netherlands. **Software Architecture** for Hard Real-Time Applications: Cyclic Executives vs. ... We begin by defining the principal application design objective which most directly determines the **software architecture** to be used for a given application: ...  
 Cited by 247 - Related articles - All 4 versions - Import into RefWorks

**Designing software architectures for usability**

J Bosch... - the 25th International Conference on Software ..., 2003 - portal.acm.org  
 ... modifiability. We should, therefore, design **software** architectures for **usability** as we design for other quality attributes [2]. This requires techniques to assess for **usability** and to improve the support for **usability** in **software architecture**. ...  
 Cited by 16 - Related articles - BL Direct - All 5 versions - Import into RefWorks

**Usability patterns in software architecture** [PDF] from psu.edu

E Folmer... - Human-computer interaction: theory and ..., 2003 - books.google.com  
**Usability** Patterns in **Software Architecture** Eelke Folmer and Jan Bosch Department of Mathematics and Computing Science University of Groningen, PO Box 800, 9700 AV the Netherlands mail@eelke.com, Jan. Bosch@cs.rug.nl Abstract Over the years the **software** ...  
 Cited by 15 - Related articles - All 10 versions - Import into RefWorks

**From single-user architectural design to PAC\*: a generic software architecture model for CSCW** [PDF] from psu.edu

G Calvary, J Couvaz... - of the SIGCHI conference on Human ..., 1997 - portal.acm.org  
 ... , frets to provide help or to log significant events to perform **usability** testing from observed behavior [24]. ... Having proxmtd the background of **software architecture** modelling and its contribution to the software design (f single-user systems, we need now to analyse its impact on WV ...  
 Cited by 129 - Related articles - BL Direct - All 11 versions - Import into RefWorks

**From system goals to software architecture** [PDF] from psu.edu

A Van Lamsweerde - Formal Methods for Software Architectures, 2003 - Springer  
 ... quality-of-service goals capture application-specific concerns about safety, security, **usability**, performance, interoperability, accuracy of **software** information with respect to what it represents in the environment, etc.; Page 4. ... From System Goals to **Software Architecture** 29 goal. ...  
 Cited by 97 - Related articles - BL Direct - All 13 versions - Import into RefWorks

**Software architecture: An executive overview** [PDF] from psu.edu

... - MELLON UNIV PITTSBURGH PA SOFTWARE ... - 1996 - Citeseer  
 ... perspective that is driven by the need to design a system that addresses needs such as concurrency, portability, evolvability, **usability**, security, etc. ... What to glean from this discussion is that a precise definition of **software architecture** is not nearly as important as the concept and ...  
 Cited by 142 - Related articles - View as HTML - All 34 versions - Import into RefWorks

**Architecting for usability: a survey** [PDF] from psu.edu

E Folmer... - Journal of systems and software, 2004 - Elsevier  
 ... Our survey shows that there are no design techniques or assessment tools that allow for design for **usability** at the architectural level. Author Keywords: **Software architecture**; **Usability**; Design for quality attributes. ... 1.3. **Software architecture** restricts **usability**. ...

Cited by 138 - Related articles - All 12 versions - Import into RefWorks

### [Beyond software architecture: creating and sustaining winning solutions](#)

L Hohmann - 2003 - portal.acm.org

... the business ramifications of portability, **usability**, configuration, upgrade and release management, security, and other architectural choices can not only lead to project failures, but ultimately to nasty lawsuits from disappointed customers. Beyond **Software Architecture** is a must ...

Cited by 64 - Related articles - All 6 versions - Import into RefWorks

### [Scenario-based software architecture reengineering](#)

PO Bengtsson... - **Software Reuse**, 1998. Proceedings. .... 1998 - ieeexplore.ieee.org

... To the best of our knowledge, few **architecture** reengineering methods have been defined. ... They spend much less effort on the **software** quality requirements that are to be fulfilled by the system. ... **usability** but, generally, no assessment of the achieved result is done. ...

Cited by 83 - Related articles - All 26 versions - Import into RefWorks

### [Improving software usability through architectural patterns](#)

N Juristo, M Lopez, AM Moreno... - Gaps Between Software ... 2003 - Citeseer

... On the other hand, the **usability** patterns in our work relate the mechanisms to be considered in a **software architecture**, addressing **usability** aspects in the early stages of the development process. ... Achieving **Usability** Through **Software Architecture**. Technical Report. ...

Cited by 31 - Related articles - View as HTML - All 11 versions - Import into RefWorks

### [Four easy pieces for assessing the usability of multimodal interaction: the CARE properties](#)

J Coutaz, L Nigay, D Salber... - Proceedings of ... 1995 - danielsalber.com

... between user preferences and system properties to show how the CARE properties interact with user modelling to predict **usability** during the design of ... for the user but may require extra processing resources from the system side or imply a specific **software architecture** (Nigay & ...

Cited by 189 - Related articles - View as HTML - All 9 versions - Import into RefWorks

### [Cameleon-rt: A software architecture reference model for distributed, migratable, and plastic user interfaces](#)

L Balme, A Demeure, N Barralon, J Coutaz... - Ambient ... 2004 - Springer

... CAMELEON-RT: A **Software Architecture** Reference Model 293 ... Applied to HCI, plasticity is the capacity of an interactive system to adapt to changes of the interactive space while preserving **usability** [2]. **Usability** is defined as a set of properties {p1, ..., pn} (eg, observability ...

Cited by 59 - Related articles - BL Direct - All 11 versions - Import into RefWorks

### [A classification and comparison of model checking software architecture techniques](#)

P Zhang, H Muccini... - Journal of Systems and Software, 2010 - Elsevier

... Goal 1 – model checking **software architecture**: the translation 4.3.1. From SA inputs to MC inputs: translation and automation 4.4. Goal 1 – summary 4.5. Goal 2 – model checking **software architecture usability** 4.5.1. Understandability 4.5.1.1. Explicit configuration 4.5.1.2. ...

Cited by 27 - Related articles - All 3 versions - Import into RefWorks

### [Comparison of scenario-based software architecture evaluation methods](#)

MA Babar... - 2004 - computer.org

... For example, assessment methods for non-traditional quality attributes (**usability**, stability etc.) are being developed. ... The **Software Architecture** Analysis Method (SAAM) first time appeared in 1993 [7]. The goals of SAAM are mainly geared to evaluate SA against the desired ...

Cited by 72 - Related articles - All 11 versions - Import into RefWorks

### [Quality characteristics for software architecture](#)

F Losavio, L Chirinos, N Lévy... - Journal of Object ... 2003 - Citeseer

... Characteristic **Usability** • Sub-characteristic ... product to enable the user to learn its application • Sub-characteristic Operability: the capability of the **software** product to ... At the architectural level, they are independent from the **architecture**, which is transparent to the users, so they ...

Cited by 47 - Related articles - View as HTML - All 7 versions - Import into RefWorks

### [The role of software architecture in constraining adaptation in component-based middleware platforms](#)

G Blair, L Blair, V Issamy, P Tuma... - Middleware 2000, 2000 - Springer

... Although work is still needed for improving the environment's **usability** (eg. ... eases the construction of middleware, contributes to **software** robustness, and fosters **software** and design ... extends to accommodate the seamless nature of the Open-ORB reflective **architecture**, ie the ...

Cited by 73 - Related articles - All 31 versions - Import into RefWorks

### [Putting non-functional requirements into software architecture](#)

X Franch... - ... of the 9th international workshop on Software ... 1998 - portal.acm.org

... Among the most widely accepted [3, 4, 13, 14, 17] we can mention: time and space efficiency, reusability, maintainability, reliability and **usability**. In our approach, we allow arbitrary identification and definition of NF-attributes. ... Page 2. 2. A Model for **Software Architecture** ...

Cited by 48 - Related articles - All 10 versions - Import into RefWorks

### [The obstacles and myths of usability and software engineering](#)

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from danielsal

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from psu.edu

A Sefah... - Communications of the ACM, 2004 - portal.acm.org

... languages, **software architecture**, component-based engineering techniques, and database integration skills are among the techniques that must be considered with HCI Web design patterns, guidelines, and methods. back to top Developing Computer-Assisted **Usability** ...

Cited by 87 - Related articles - BL Direct - All 5 versions - Import into RefWorks

### [On non-functional requirements in software engineering](#)

L Chung... - Conceptual Modeling: Foundations and ... 2009 - Springer

... Colloquially speaking, NFRs have been referred to as "ilities" (eg, **usability**) or "ities" (eg, integrity), ie, words ending with the string "-ility" ... More details on the NFR Framework will be described further in Section 4. In the area of **Software Architecture**, one frequently encountered ...

Cited by 1440 - Related articles - BL Direct - All 8 versions - Import into RefWorks

### [Software architecture reconstruction](#)

RL Krikhaar - 1999 - Citeseer

Page 1. René L. Krikhaar Softw are **Architecture** Reconstruction René L. Krikhaar 1999 **Software Architecture** Reconstruction Afhankelijk van rugdikte Page 2. **Software Architecture** Reconstruction Page 3. ... 172 Page 27. Page 28. Chapter 1 **Software Architecture** ...

Cited by 72 - Related articles - View as HTML - All 14 versions - Import into RefWorks

### [Coming attractions in software architecture](#)

P Clements - wpdrt, 1997 - computer.org

... For instance, a simulator could be used to analyze for **usability** by let ... to UniCon; automatic generation of wrappers for standard distrib- uted **software** component communication ... theorem-proving verification, such as provided by Modechart Communicating an **architecture** to a ...

Cited by 52 - Related articles - All 15 versions - Import into RefWorks

### [On the definition of software system architecture](#)

C Gacek, A Abd-Allah, B Clark... - ... on Architectures for Software ... 1995 - profinit.eu

... The user will be interested at the architecting stage in the impact of the **software** structure on performance, **usability**, and compliance with other system attribute requirements. As with architectures of buildings, users also need to relate the **architecture** to their usage scenarios. ...

Cited by 69 - Related articles - View as HTML - All 9 versions - Import into RefWorks

### [Software architecture for language engineering](#)

H Cunningham - 2000 - Citeseer

Page 1. **Software Architecture** for Language Engineering Hamish Cunningham ... Abstract This thesis defines the boundaries of **Software Architecture** for Language Engineering (SALE), an area formed by the intersection of human language computation and **software** engineering. ...

Cited by 77 - Related articles - View as HTML - All 12 versions - Import into RefWorks

### [A basis for analyzing software architecture analysis methods](#)

R Kazman, L Bass, M Klein, T Lattanze... - Software Quality ... 2005 - Springer

... attribute research communities, we have created standard characterizations for performance, availability, **usability**, testability, se ... **Software** fault is refined in the utility tree of figure 1 into "Unlock ... for an analyst or architect to precisely analyze whether the **architecture** satisfies the ...

Cited by 38 - Related articles - BL Direct - All 10 versions - Import into RefWorks

### [A software architecture supporting networked sensors](#)

J Hill - 2000 - Citeseer

Page 1. A **Software Architecture** Supporting Networked Sensors by Jason Hill Research Project ... Date Kristofer Pister, Second Reader Date Fall 2000 Page 2. A **Software Architecture** Supporting Networked Sensors Copyright Fall 2000 by Jason Hill Page 3. iii Acknowledgements ...

Cited by 69 - Related articles - View as HTML - All 15 versions - Import into RefWorks

### [Applying general usability scenarios to the design of the software architecture of a collaborative workspace](#)

RJ Adams, L Bass... - ... Frameworks for HCI/HCD and Software ... 2005 - Citeseer

Abstract: Architecturally-sensitive **usability** scenarios are important **usability** concerns that require early consideration in **software** design so that architectural support can render them easy and cost-effective to implement. Examples include providing the ability to cancel a command, ...

Cited by 8 - Related articles - View as HTML - All 3 versions - Import into RefWorks

### [Essential software architecture](#)

I Gorton... - 2006 - bib.tiera.ru

Page 1. Essential **Software Architecture** Page 2. Ian Gorton Essential **Software Architecture** ... In the years that I have known Ian, he has been an inspirational educa- tor, a pragmatic and decisive **software** architect, and an idealistic **software architecture** researcher. ...

Cited by 71 - Related articles - View as HTML - All 8 versions - Import into RefWorks

### [Analysing the impact of usability on software design](#)

N Juristo, AM Moreno... - ... of Systems and Software, 2007 - Elsevier

... time. Keywords: **Software usability**; **Software** design. Article Outline. 1. Introduction ... 1. Introduction. **Software usability** is a quality attribute listed in a number of classifications ([IEEE, 1998], [ISO/IEC, 1991] and [Boehm, 1978]). Although it is ...

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from profinit.eu

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from tiera.ru



Cited by 30 - Related articles - All 2 versions - Import into RefWorks

**Aspect oriented software architecture: a structural perspective**

A Navasa, MA Pérez, JM Muñilo... - Workshop on Early Aspects ... 2002 - Citeseer  
... development process. These include: increase in productivity, re-usability and adaptability. Thus ... applications [Gru00]. It seems reasonable to observe **software architecture** from the aspect-oriented point of view due to several reasons. On ...  
Cited by 37 - Related articles - View as HTML - All 8 versions - Import into RefWorks

[PDF] from psu.edu

**Utility and usability: research issues and development contexts**

J Grudin - Interacting with computers, 1992 - Elsevier  
... Isolating aspects of **usability** promotes flexibility - interface code and text that are distinct from other **software** are more easily ... The question underlying these considerations addresses the **software architecture**, the product of development. If **software** is designed to separate form ...  
Cited by 91 - Related articles - Import into RefWorks

**Software architecture and software configuration management**

B Westfechtel... - Software Configuration Management, 2003 - Springer  
... While some of these may be organized according to the **software architecture** (eg mod-ule ... languages strive for a high semantic level which allows one to analyze **software architectures** for ... To provide for re-usability, SCM systems abstract from the contents of **software objects** ...  
Cited by 35 - Related articles - BL Direct - All 17 versions - Import into RefWorks

[PDF] from psu.edu

**A System Software Architecture for High End Computing**

DS Greenberg, R Brightwell, LA Fisk, A McCabe... - 1997 - computer.org  
... system **software**. At Sandia National Laboratories we have developed, with our vendors, a new system **architecture** for high- end computing. Highest performance is achieved by providing applications with a light-weight interface to a collection of processing nodes. **Usability** is ...  
Cited by 52 - Related articles - All 10 versions - Import into RefWorks

[PDF] from psu.edu

**A multitouch software architecture**

F Echter... - Proceedings of the 5th Nordic conference on ... 2008 - portal.acm.org  
... their requirements, we are re- fining the key components of the **architecture**, especially the ... As the **usability** and success of any such framework depend on usage and feedback by ... 5. DISCUSSION In terms of interoperability with other **software**, special consider- ations apply with ...  
Cited by 33 - Related articles - All 7 versions - Import into RefWorks

[PDF] from psu.edu

**Software architecture for processing clusters based on I2O**

J Gutleber... - Cluster Computing, 2002 - Springer  
... Peer-to-peer measurements are, how- ever, not sufficient to demonstrate a toolkits **usability**. There ... Event based processing allows us to Page 9. **SOFTWARE ARCHITECTURE FOR PROCESSING CLUSTERS BASED ON I2O** 63 build ...  
Cited by 49 - Related articles - BL Direct - All 4 versions - Import into RefWorks

**An investigation of a method for identifying a software architecture candidate with respect to quality attributes**

M Svahnberg... - Empirical Software Engineering, 2005 - Springer  
... METHOD FOR IDENTIFYING A **SOFTWARE ARCHITECTURE** CANDIDATE 165 ... of groups of agreeing people and the sizes of these groups for each **architecture** candidate and ... For example, for Model-View-Controller and **Usability** five of the participants were in agreement, two ...  
Cited by 25 - Related articles - All 16 versions - Import into RefWorks

[PDF] from ksu.edu

**Software for use: a practical guide to the models and methods of usage-centered design**

LL Constantine... - 1999 - portal.acm.org  
... **architecture**. In this book, they present the models and methods of a revolutionary approach to **software** that will help programmers deliver more usable **software-software** that will enable users to accomplish their tasks with greater ease and efficiency. Recognizing **usability** as ...  
Cited by 580 - Related articles - All 6 versions - Import into RefWorks

**On the relationship of software architecture to software component technology**

K Wallnau, J Stafford, S Hissam... - ECOOP2001 WCOP ... 2001 - Citeseer  
... At present, five quality attributes (security, performance, **usability**, modifiability, availability) have been character- ized in this way. ... contrast to the previous illustration, the metatypes we now define are not grounded in component technology, but in **software architecture** technology ...  
Cited by 38 - Related articles - View as HTML - All 9 versions - Import into RefWorks

[PDF] from psu.edu

**Remote evaluation: the network as an extension of the usability laboratory**

HR Hartson, JC Castillo, J Kelso... - Proceedings of the ... 1996 - portal.acm.org  
... First, to simulate clicking of the **software** "Report CF" button, user subjects pushed the space bar on ... screen detail and their effect on the ability to discern useful information about **usability** problems ... are using an expert system for landscape **architecture** as part of a very large scale ...  
Cited by 145 - Related articles - All 3 versions - Import into RefWorks

**Experiences in assessing product family software architecture for evolution**

A Maccani - Software Engineering, 2002. ICSE 2002. .... 2002 - ieeexplore.ieee.org

[PDF] from ua.ac.be

... choices done by the manufacturer, usually for the sake of simpler design, higher **usability** or better ... When we started the assessment, most of the system **software** had not yet been written ... context, the fit goal of the assessment was to improve the quality of the existing **architecture**. ...  
Cited by 41 - Related articles - BL Direct - All 9 versions - Import into RefWorks

**Cognitive support features for software development tools**

JE Robbins - 1999 - argouml.tigris.org  
... 30 Table 2-5: **Usability** guidelines from Shneiderman (1998) 30 ... no. 6. June 1996. pp. 390-406. A significant revision and extension of the ICSE'95 paper. Extending Design Environments to **Software Architecture** Design. Jason E. Robbins, David M. Hilbert, David F. Redmiles. ...  
Cited by 32 - Related articles - View as HTML - All 4 versions - Import into RefWorks

[PDF] from tigris.org

**Software Architecture: Foundation of a Software Component Marketplace**

EJ Whitehead Jr, JE Robbins, N Medvodic... - ... for Software Systems, 1995 - Citeseer  
... has multiple sales models, ranging from single sale, single-user (PC **software** model), to ... Also, an **architecture** should ideally provide sup- port for differing sales models, for example ... Unlike traditional **usability** of a product interface where the only significant interface is between ...  
Cited by 28 - Related articles - View as HTML - All 18 versions - Import into RefWorks

[PDF] from psu.edu

**Quality vs. quantity: Comparing evaluation methods in a usability-focused software architecture modification task**

E Golden, BE John... - ... Symposium on Empirical Software ... 2005 - computer.org  
... In this paper, first we justify the importance of **usability** as it relates to **software architecture**, and briefly discuss **Usability-Supporting Architecture Patterns (USAPs)**, which address the handling of specific **usability** concerns in the **software architecture** design process. ...  
Cited by 7 - Related articles - All 10 versions - Import into RefWorks

[PDF] from psu.edu

**A concern-oriented approach to software architecture**

MM Kandé - 2003 - infoscience.epfl.ch  
... A CONCERN-ORIENTED APPROACH TO **SOFTWARE ARCHITECTURE** ... This dissertation presents a new approach to **software architecture** that is suitable for supporting concern-oriented development and documentation of architectures for **software- intensive** systems. ...  
Cited by 50 - Related articles - All 11 versions - Import into RefWorks

[PDF] from epfl.ch

**Controlshell: A software architecture for complex electromechanical systems**

SA Schneider, VW Chen... - ... Journal of Robotics ... 1998 - jir.sagepub.com  
... The International Journal of Robotics Research Stanley A. Schneider, Vincent W. Chen, Gerardo Pardo-Castellote and Howard H. Wang ControlShell: A **Software Architecture** for Complex Electromechanical Systems Published by: ... **Software Architecture** for Complex ...  
Cited by 45 - Related articles - BL Direct - Import into RefWorks

**A method to elicit architecturally sensitive usability requirements: its integration into a software development process**

T Rafia, PN Robillard... - Software Quality Journal, 2007 - Springer  
... 2.2 **Software architecture** that supports **usability** (STATUS) project ... Once the **usability** requirements have been identified, the next step is to find mechanisms that might be incorporated into the **software architecture** to improve the **usability** of the system (Fig. ...  
Cited by 19 - Related articles - BL Direct - All 4 versions - Import into RefWorks

**Process agility and software usability: Toward lightweight usage-centered design**

LL Constantine - Information Age, 2002 - informatix.ida.liu.se  
... True **usability** testing requires repeated testing with numbers of users under controlled settings ... refinement in small increments is the absence of any comprehensive overview of the entire **architecture**. For internal elements of the **software**, this shortcoming is not fatal, because the ...  
Cited by 55 - Related articles - All 10 versions - Import into RefWorks

[PDF] from liu.se

**Scenario-based software architecture evaluation methods: An overview**

MT Ionita, DK Hammer... - ICSE/SARA, 2002 - Citeseer  
... as to provide a basis for an informed decision process with regards to **architecture**. ... attributes, which need to be considered among the tradeoffs when a **software** system is ... with respect to architectural quality attributes like modifiability, performance, availability, **usability**, and so on ...  
Cited by 35 - Related articles - View as HTML - All 6 versions - Import into RefWorks

[PDF] from psu.edu

**An empirical study of groupware support for distributed software architecture evaluation process**

MA Babar, B Kitchenham, L Zhu, I Gorton... - ... of Systems and Software, 2006 - Elsevier  
... requires expertise and knowledge of different quality attribute experts such as performance engineers and **usability** specialists ... Based on these previous results, we are developing the concept of groupware supported distributed **software architecture** evaluation processes, which ...  
Cited by 41 - Related articles - All 5 versions - Import into RefWorks

[PDF] from ntnu.no

**An ontology of architectural design decisions in software intensive systems**

P Kruchten - 2nd Groningen Workshop on Software Variability, 2004 - Citeseer  
... Example: "Use GIS Mapinfo" Categories: politics, **usability**, safety, COTS 4.7. Cost ... 9. References Bosch, J. (2004, May). **Software Architecture: the Next Step**. Paper presented at the First European Workshop on **Software Architecture** (EWSA 2004), St Andrews, Scotland. ...

[PDF] from psu.edu

Cited by 108 - Related articles - View as HTML - All 5 versions - Import into RefWorks

### book Usability for the Web: designing Web sites that work

T Brinck, D Gergle, SD Wood... - 2002 - lavoisier.fr

... task analysis chapter. information **architecture** mockups and prototypes chapter. page layout chapter. envisioning design production chapter. writing for the web chapter. design elements chapter. **usability in software** development launch chapter. ...

Cited by 151 - Related articles - Cached - All 7 versions - Import into RefWorks

### citation Investigating the Relationship between Usability and Software Architecture

E Folmer, J Gurp... - **Software process improvement and practice**, Wiley, 2003

Cited by 7 - Related articles - Import into RefWorks

### Measuring the usability of software components

MF Bertoa, JM Troya... - **Journal of Systems and Software**, 2006 - Elsevier

... Permissions & Reprints. Measuring the **usability of software** components. ... At least three measurable concepts are closely related to **software** component **Usability**: the Quality of the Documentation, the Complexity of the Problem and the Complexity of the Solution (or Design). ...

Cited by 33 - Related articles - All 9 versions - Import into RefWorks

### Principles for a usability-oriented pattern language

MJ Mahemoff... - ozchi, 1998 - computer.org

... has elements that can be iterated. Patterns of **software architecture**, such as Gamma et al.'s patterns, could be used in this way to support functions or user-interface features which are proposed by **usability**-oriented patterns. ...

Cited by 52 - Related articles - All 14 versions - Import into RefWorks

### Mining patterns to support software architecture evaluation

L Zhu, MA Babar... - 2004 - computer.org

... 1), and has recently emerged as an important quality assurance technique known as **software architecture** (SA) evaluation. It has been shown that SA constrains the achievement of various quality attributes (such as performance, security, maintainability and **usability**) in a ...

Cited by 26 - Related articles - All 9 versions - Import into RefWorks

### paper A reusable operational software architecture for advanced robotics

C Kapoor... - COURSES AND LECTURES-INTERNATIONAL ..., 1998 - Citeseer

... Chetan Kapoor 1996 Page 2. A Reusable Operational **Software Architecture** for Advanced Robotics by Chetan Kapoor, MS Dissertation ... December, 1996 Page 3. A Reusable Operational **Software Architecture** for Advanced Robotics Approved by Dissertation Committee: Page 4. ...

Cited by 43 - Related articles - View as HTML - BL Direct - All 7 versions - Import into RefWorks

### paper Towards modeling non-functional requirements in software architecture

L Xu, H Ziv, D Richardson... - **Early Aspects**, 2005 - Citeseer

... NFRs to **software** architectures the same or different? Are techniques for analysis and testing of architectures against FRs and NFRs the same or different? and so on. In particular, analyzing the **architecture** against a number of critical NFRs, such as security, **usability**, integrity ...

Cited by 21 - Related articles - View as HTML - All 7 versions - Import into RefWorks

### paper A family of software architecture implementation frameworks

N Medvidovic, NR Mehta... - on **Software Architecture** ..., 2002 - Citeseer

Page 1. A Family of **Software Architecture** Implementation Frameworks Nenad Medvidovic Nikunj Mehta Manjiv Mikic-Rakic Computer Science Department Henry Salvatori Computer Science Center 300 University of Southern ...

Cited by 34 - Related articles - View as HTML - All 9 versions - Import into RefWorks

### Software architecture modeling for user interfaces

J Coutaz - Wiley Online Library

... Where scenarios are used to illustrate aspects of **usability** along with their corresponding architectural patterns. In summary, **software** designers must consider multiple perspectives on architectural design. As a result, there is no such thing as "the **software architecture** of a ...

Cited by 26 - Related articles - All 12 versions - Import into RefWorks

### An explicit definition of connectors for component-based software architecture

M Oussalah, A Smeda... - 2004 - computer.org

... In section 3 we present an overview of a multi-paradigm approach to describe the complex systems called Component-Object based **Software Architecture** (CO-SA). ... Hard to use: the **usability** of connectors (in the form of properties of components) is proved to be difficult. ...

Cited by 29 - Related articles - All 5 versions - Import into RefWorks

### book Large-scale software architecture: a practical guide using UML

J Garland... - 2003 - books.google.com

Page 1. Large-Scale **Software Architecture** A Practical Guide using UML Jeff Garland CrystalClear **Software** Inc. Richard Anthony Object Computing Inc. Page 2. Page 3. Large-Scale **Software Architecture** Page 4. Page 5. ...

Cited by 41 - Related articles - All 8 versions - Import into RefWorks

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from psu.edu

[PDF] from psu.edu

### A CASE tool for software architecture design

K Ng, J Kramer... - **Automated Software Engineering**, 1996 - Springer

... the Architect's Assistant supports a compositional approach to program development in which the **software architecture** plays a central role throughout the **software** life-cycle ... Conscious effort has been made to maximise **usability** and efficiency, primarily by enhancing the level ...

Cited by 28 - Related articles - All 4 versions - Import into RefWorks

### Usability engineering turns 10

KA Butler - interactions, 1996 - portal.acm.org

... **Usability** is a Quality of **Software** The user interface is critically important for the success of an interactive computer system. ... It can also be used to improve the **software** process itself because **usability** measurements indicate how well a user interface design process is working. ...

Cited by 70 - Related articles - BL Direct - All 2 versions - Import into RefWorks

### GATE: A framework and graphical development environment for robust NLP tools and applications

DH Cunningham, DD Maynard, DK Boricheva... - 2002 - eprints.aktors.org

... In addition, it promotes robustness, re-**usability**, and scalability as important principles that help with the construction of practical NLP systems. 8 Conclusions ... H. Cunningham. 2000. **Software Architecture** for Language Engineering. Ph.D. thesis, University of Sheffield. ...

Cited by 1060 - Related articles - All 3 versions - Import into RefWorks

### Generation of distributed system test-beds from high-level software architecture descriptions

J Grundy, Y Cai... - ase, 2001 - computer.org

... Proceedings of the 16th Annual International Conference on Automated **Software Engineering** (ASE 2001) ... Song, W., Tan, CJ Distributed and scalable XML document processing **architecture** for E ... 7. Green, TRG and Petre, M, **Usability** analysis of visual programming environments ...

Cited by 30 - Related articles - All 20 versions - Import into RefWorks

### paper Software architecture for a robot teleoperation system

A Alonso, B Alvarez, JA Pastor... - 4th IFAC Workshop on ..., 1997 - dit.upm.es

... **Usability**: The system will be used by operators with no background in computing. Hence it must be easy to use and adapted to the skills and common practices in their domain. ... Local Area Network Tools **Software architecture** for a robot teleoperation system ...

Cited by 23 - Related articles - View as HTML - Import into RefWorks

### book User interface management systems

GE Pfaff - 1985 - portal.acm.org

... **software** application **architecture**, Proceedings of the 12th international conference on **Software engineering**, p.212-220, March 26-30, 1990, Nice, France. Tamer Rafia , Pierre N. Robillard , Michel Desmarais, A method to elicit architecturally sensitive **usability** requirements: its ...

Cited by 391 - Related articles - Import into RefWorks

### A design space for multimodal systems: concurrent processing and data fusion

L Nigay... - Proceedings of the INTERACT'93 and CHI'93 ..., 1993 - portal.acm.org

... Given that the command language barrier is surpassed, **usability** can be further tested by establishing whether particular modalities are adequate for expressing a given command.

For instance, if a command has a ... **SOFTWARE ARCHITECTURE FOR** ...

Cited by 266 - Related articles - All 12 versions - Import into RefWorks

### Guidelines for eliciting usability functionalities

N Juristo, A Moreno... - Transactions on **Software** ..., 2007 - computer.org

... 44]. Bass et al. [2], [3] have used a bottom-up approach based on fieldwork observation to describe a set of scenarios representing **usability** issues that have an effect on the **software architecture**. We 1 have decomposed ...

Cited by 55 - Related articles - BL Direct - All 5 versions - Import into RefWorks

### Characterizing software architecture changes: A systematic review

BJ Williams... - Information and **Software Technology**, 2010 - Elsevier

... Permissions & Reprints. Characterizing **software architecture** changes: A systematic review. ... Abstract. With today's ever increasing demands on **software**, **software** developers must produce **software** that can be changed without the risk of degrading the **software architecture**. ...

Cited by 36 - Related articles - All 8 versions - Import into RefWorks

### Component-based software engineering—new challenges in software development

I Crnkovic - **Software Focus**, 2001 - Wiley Online Library

... This applies to both functional and non-functional requirements. 4 Conflict between **usability** and reusability. ... Discussion of these is beyond the scope of this article and the relation between **software architecture** and CBD is discussed in the following. ...

Cited by 99 - Related articles - BL Direct - All 6 versions - Import into RefWorks

### Accommodating usability driven changes in existing software architecture

T Rafia, R Oketokoun, A Wikik... - ..., on **Software** ..., 2004 - actaexpress.com

... This assumption is now challenged since **usability** is strongly related to the internal components of the system and it must be considered when designing the rest of the system [3]. Decisions in **software architecture** can severely compromise the **usability** of the final system. ...

[PDF] from aktors.or

[PDF] from auckland

[PSI] from upm.es

[PDF] from psu.edu

[PDF] from msstate.

[PDF] from psu.edu

[PDF] from tripod.co



Cited by 5 - Related articles - All 6 versions - Import into RefWorks

[A survey of architecture description languages](#)

PC Clements - ... of the 8th international workshop on **software** ... 1996 - portal.acm.org  
... **Architecture** analysis support: What support is provided by the ADL for analyzing **architecture**-level information in order to predict ... Analyzing for portability: hardware independence, **software** independence ... Analyzing for **usability**: understandability, ease of learning, ...  
Cited by 287 - Related articles - All 22 versions - Import into RefWorks

[PDF] from javeriana

[citation] **Usability and software architecture**

BEJL Bass, BE John, N Juristo... - Behaviour & Information ..., 2001  
Cited by 6 - Related articles - Import into RefWorks

[Capturing and using software architecture knowledge for architecture-based software development](#)

MA Babar, II Gorton... - 2005 - computer.org  
... 2000. [40] John, BE, et al., "Bringing **Usability** Concerns to the Design of **Software Architecture**," Proc. of the 9th IFIP Working Conference on Engineering for Human-Computer Interaction. 2004. [41] "Hipergate - Open Source CRM and Groupware", http://www.hipergate.com. ...  
Cited by 22 - Related articles - All 3 versions - Import into RefWorks

[Moving towards quality attribute driven software architecture reconstruction](#)

C Stoermer, L O'Brien... - 2003 - computer.org  
... For example performance models can be fairly formal (queuing models) while **usability** models try to model user satisfaction, which ... The **architecture** tactics are not free to select. In the design process the **software** architect has to select the appropriate tactic to satisfy the required ...  
Cited by 26 - Related articles - All 6 versions - Import into RefWorks

[PDF] from vu.nl

[Identifying quality-requirement conflicts](#)

B Boehm... - Software, IEEE, 1996 - ieeexplore.ieee.org  
... Requirements big., T Press, T O'Leary, C. Alford, 1 90j, IEEE SOFTWARE Page 6. Attribute ... MARCH 1996 Page 7. Primary **Architecture** Attribute Strategy Assurance Input checking ... Reinforcement Conflicts Comments Interoperability, **usability** Coadschedule performance ...  
Cited by 262 - Related articles - BL Direct - All 16 versions - Import into RefWorks

[PDF] from unp.edu.

[Clarifying the relationship between software architecture and usability](#)

N Juristo... - Sixteenth International Conference on Software ..., 2004 - Citeseer  
This paper examines in a problem posed recently concerning the relationship between **software** system **usability** and **architecture**. Here, we try to empirically clarify this relationship, focusing on the concept of **architecture**-sensitive **usability** mechanism. This concept represents ...  
Cited by 4 - Related articles - Cached - All 2 versions - Import into RefWorks

[Integrating usability techniques into software development](#)

J Anderson, F Fleak, K Garrity... - Software, IEEE, 2001 - ieeexplore.ieee.org  
... The **software** development process we developed might seem a generic solution. ... model as closely as possible within the constraints of the information and technical **architecture**. The **usability** evaluator has primary responsibility for testing the product design, analyzing and ...  
Cited by 45 - Related articles - BL Direct - All 9 versions - Import into RefWorks

[PDF] from aau.dk

[book] **SACAM: The software architecture comparison analysis method**

... -MELLON UNIV PITTSBURGH PA SOFTWARE ... - 2003 - Citeseer  
... quality attribute. A collection of tactics for a variety of quality attributes, such as modifiability, performance, **usability**, testability, and availability, are presented in the book **Software Architecture in Practice** [Bass 03]. The set of ...  
Cited by 25 - Related articles - All 17 versions - Import into RefWorks

[PDF] from psu.edu

[book] **Quality attributes in software architecture design**

L Lundberg, J Bosch, D Häggander... - Conference on Software ..., 1999 - Citeseer  
... As part of our future work, we intend to incorporate other attributes, eg reusability, **usability** and availability. ... However, in ATAM there are no guidelines on how to modify the **software architecture**. Figure 7: Outline of the architectural design method ...  
Cited by 26 - Related articles - View as HTML - All 11 versions - Import into RefWorks

[PDF] from psu.edu

[book] **Illuminating the fundamental contributors to Software Architecture Quality**

... -MELLON UNIV PITTSBURGH PA SOFTWARE ... - 2002 - Citeseer  
... One of the key principles that direct our work is that the quality-attribute requirements (such as performance, security, modifiability, reliability, and **usability**) exert a dominant influence on the "shape" of a **software architecture**. ...  
Cited by 32 - Related articles - View as HTML - All 11 versions - Import into RefWorks

[PDF] from psu.edu

[Reconciling usability and interactive system architecture using patterns](#)

A Seffah, T Mohamed, H Habieb-Mammar... - Systems and Software, 2008 - Elsevier  
Traditional interactive system architectures such as MVC [Goldberg, A., 1984, Smaltalk-80: The Interactive Programming Environment, Addison-Wesley Publ.] and PAC [Coutaz, J., 1987, PAC, an implementation model for dialog design. In: Interact'87, Stuttgart, September ...

Cited by 12 - Related articles - All 2 versions - Import into RefWorks

[book] **Usability and open source software**

DM Nichols, M Twidale... - 2002 - Citeseer  
... Although there are few formal studies of open source **usability** there are several suggestions that open source **software usability** is a significant issue (Behlendorf, 1999; Raymond, 1999; Manes, 2002; Nichols et al., 2001; Thomas 2002; Frishberg et al., 2002): If this [desktop and ...  
Cited by 146 - Related articles - View as HTML - All 38 versions - Import into RefWorks

[PDF] from psu.edu

[Helping software architects design for usability](#)

E Golden - Proceedings of the 1st ACM SIGCHI symposium on ..., 2009 - portal.acm.org  
... the architectural ramifications of **usability** requirements, then **Usability**-Supporting Architectural Patterns (USAPs) will help to bridge the gap between UI designers and **software** engineers to produce **software architecture** solutions that successfully address **usability** requirements ...  
Cited by 6 - Related articles - Import into RefWorks

[Software technology in an automotive company-major challenges](#)

K Grimm - 2003 - computer.org  
... This sub-process aims at the seamless integration of all **usability** aspects into the entire **software** development process [1]. Major elements of ... As the **software architecture** in a vehicle also depends on the mechanical and the electrical **architecture** there is also a mutual ...  
Cited by 85 - Related articles - BL Direct - All 6 versions - Import into RefWorks

[ROSATEA: International workshop on the role of software architecture in analysis e \(and\) testing](#)

D Richardson... - ACM SIGSOFT Software Engineering ..., 1999 - portal.acm.org  
... This paper outlines some ways in which architectural description languages need to be designed to increase their **usability**, acceptability, and ... **Software architecture** description languages provide a means to formally describe **software** systems at a high level of abstraction. ...  
Cited by 25 - Related articles - Import into RefWorks

Create email alert



software architecture usability

[Go to Google Home](#) - [About Google](#) - [About Google Scholar](#)

©2011 Google

[Guidelines for eliciting usability functionalities](#)

N Juristo, A Moreno... - Transactions on Software ... 2007 - computer.org  
 ... These details are important as they have an impact on this mechanism's design ... Evidently, the use of **usability patterns** and any other artifact for improving software system **usability** calls for a lot of user involvement throughout the development process. ...  
 Cited by 55 - Related articles - BL Direct - All 5 versions - Import into RefWorks

[Bringing usability concerns to the design of software architecture](#)

BE John, L Bass, MI Sanchez-Segura... - Computer Interaction and ... 2005 - Springer  
 ... We then discuss why we are focusing on forces and why the forces that come from prior design decisions play a special role in software creation. In section 4, we describe our template for these patterns and illustrate it with one of the usability scenarios previously identified by ...  
 Cited by 28 - Related articles - BL Direct - All 13 versions - Import into RefWorks

[Scenario-based assessment of software architecture usability](#)

E Folmer, J Van Gurp... - Bridging the Gaps Between Software ... 2003 - Citeseer  
 ... **Usability patterns** The term **usability pattern** refers to a technique or mechanism that can be applied to the design of the architecture of a software system in order to address a need identified by a **usability** property at the requirements Figure 2: **Usability framework** 63 Page 64. ...  
 Cited by 22 - Related articles - View as HTML - All 18 versions - Import into RefWorks

[Empowering software engineers in human-centered design](#)

A Saffah... - International conference on Software ... 2003 - portal.acm.org  
 ... transferring, by means of software development tools, the design knowledge of human factors and user interface designers to software engineers unfamiliar with usability engineering. Within our approach, we have been using three different categories of patterns: HCI (Human ...  
 Cited by 19 - Related articles - BL Direct - All 6 versions - Import into RefWorks

[The usability engineering lifecycle: a practitioner's handbook for user interface design](#)

DJ Mayhew - 1999 - books.google.com  
 ... According to Karat and Dayton (1995), "In most cases of the design and development of commercial software, usability is not dealt with at the same level as other aspects of software engineering (eg, clear usability objectives are not set, resources for appropriate activities are ...  
 Cited by 544 - Related articles - BL Direct - All 4 versions - Import into RefWorks

[Patterns in property specifications for finite-state verification](#)

MB Dwyer, GS Avrunin... - 1999 - computer.org  
 ... Thus a pattern system for proper ties should mirror this view to enhance usability. ... In fact, the frequencies of the patterns, 416 Proceedings of the 21st International Conference on Software Engineering (ICSE'99) 0270-5257/99 \$ 10.00 © 1999 ACM Page 7. ...  
 Cited by 698 - Related articles - BL Direct - All 19 versions - Import into RefWorks

[Designing interfaces](#)

J Tidwell - 2005 - portal.acm.org  
 ... Being in web design, I know that the mobile world is really taking off, so using design patterns from mobile devices and ... on various devices, and this book really illuminates those areas if you haven't had much experience with software development or mobile usability testing. ...  
 Cited by 215 - Related articles - All 9 versions - Import into RefWorks

[Architectural styles and the design of network-based software architectures](#)

RT Fielding - 2000 - Citeseer  
 ... Architectural Styles and the Design of Network-based Software Architectures DISSERTATION ... with the categorization of software designs and the development of design methodologies, but has rarely been able to objectively evaluate the impact of various design choices on ...  
 Cited by 1699 - Related articles - View as HTML - All 76 versions - Import into RefWorks

[Principles for a usability-oriented pattern language](#)

MJ Mahemoff... - ozchi, 1998 - computer.org  
 ... Furthermore, a vocabulary of patterns which consider both interaction and software design would be a boon to projects of an interdisciplinary nature. To identify patterns which promote usability, it is necessary to understand exactly what we mean by usability. ...  
 Cited by 52 - Related articles - All 14 versions - Import into RefWorks

[Object-oriented software engineering](#)

T Lethbridge... - 2001 - sut.ac.th  
 ... to employ additional design patterns 246 6.15 Difficulties and risks when using design patterns 250 6.16 ... users 258 7.3 The basics of user interface design 258 7.4 Usability principles 262 ... Summary 307 8.7 For more information 307 9 Architecting and designing software 309 9.1 ...  
 Cited by 122 - Related articles - All 5 versions - Import into RefWorks

[A software architectural view of usability patterns](#)

X Ferré, N Jusisto, AM Moreno... - Proceedings of INTERACT ... 2003 - Citeseer  
 ... so on), that are needed for the software product being developed, and providing software developers with design solutions that address such usability properties. The design solutions proposed are shaped in the form of patterns, which express common usability heuristics of the ...  
 Cited by 9 - Related articles - View as HTML - All 9 versions - Import into RefWorks

[Deriving software usage patterns from log files](#)

... of Technology, Graphics, Visualization and Usability ... - 1993 - Citeseer  
 ... if all operations were exercised equally, with no preference suggested by the usability of the ... on log file analysis can result from identifying high-level actions and viewing software usage in ... The technique described in this paper has proven useful in describing patterns in student ...  
 Cited by 25 - Related articles - View as HTML - All 12 versions - Import into RefWorks

[The value of a usability-supporting architectural pattern in software architecture design: a controlled experiment](#)

E Golden, BE John... - International conference on Software ... 2005 - portal.acm.org  
 ... ABSTRACT Design patterns have been claimed to facilitate modification and improve understanding in software design. A controlled experiment was performed to assess the usefulness of portions of a Usability-Supporting Architectural Pattern (USAP) in modifying the ...  
 Cited by 25 - Related articles - BL Direct - All 11 versions - Import into RefWorks

[Application of Software design patterns to DSP library design](#)

P Astrom, S Johansson... - System Synthesis, 2001. ... 2001 - ieeexplore.ieee.org  
 ... All software patterns cannot be used in hardware design, however they can provide insight into how ... We have shown by an example applying the described patterns that a C++ design is ... Several important properties of a library like usability, reusability, ease of use and structure ...  
 Cited by 15 - Related articles - All 9 versions - Import into RefWorks

[Organizational Patterns](#)

J Caplan - Enterprise Information Systems VI, 2006 - Springer  
 ... Does success depend on usability of the software? Or does your company base success on sales? ... Organizational patterns are a guide to shaping the organizational structures that bode for success. The software architecture echoes that same structure by Conway's Law. ...  
 Cited by 181 - Related articles - All 6 versions - Import into RefWorks

[Software architecture analysis of usability](#)

E Folmer, J van Gurp... - Engineering Human Computer ... 2005 - Springer  
 ... 2.3 Architecture Sensitive Usability Patterns A number of usability patterns have been identified that should be applied during the design of a system's software architecture, rather than during the detailed design stage. This set ...  
 Cited by 34 - Related articles - BL Direct - All 27 versions - Import into RefWorks

[Designing software architectures for usability](#)

J Bosch... - the 25th International Conference on Software ... 2003 - portal.acm.org  
 ... o use-case map based assessment o integrated usability assessment Improving Software Architectures for Usability o usability patterns o selecting design solutions • Case studies: examples and experiences • Concluding remarks ...  
 Cited by 16 - Related articles - BL Direct - All 5 versions - Import into RefWorks

[Property specification patterns for finite-state verification](#)

MB Dwyer, GS Avrunin... - Formal methods in software ... 1998 - portal.acm.org  
 ... sis and modeling information [16], software process and organizational structures [5], and curricula for educating software developers [22] ... patterns constrain the order of states/events [eg, the ... Thus a pattern system for properties should mirror this view to enhance usability. ...  
 Cited by 283 - Related articles - All 13 versions - Import into RefWorks

[From non-functional requirements to design through patterns](#)

D Gross... - Requirements Engineering, 2001 - Springer  
 ... functional requirements, also called quality attributes (eg [1,2]). These are requirements such as reliability, usability, maintainability, cost ... [6] to support architectural design [7,8] and to deal with change [9]. In the software design area, the concept of design patterns has been ...  
 Cited by 131 - Related articles - BL Direct - All 40 versions - Import into RefWorks

[An Ontology-Based Metamodel for Software Patterns](#)

S Henninger... - 18th Int. Conf. on Software Engineering ... 2006 - Citeseer  
 ... LNCS 2254, Springer, 2001, pp. 141-155. [14] S. Henninger, P. Ashokkumar, "An Ontology-Based Infrastructure for Usability Design Patterns." Proc. Semantic Web Enabled Software Engineering (SWESE), Galway, Ireland, pp. 41-55, 2005. ...  
 Cited by 13 - Related articles - View as HTML - All 10 versions - Import into RefWorks

[A framework for capturing the relationship between usability and software architecture](#)

E Folmer, J Van Gurp... - Software Process: ... 2003 - Wiley Online Library  
 ... However, we want to have this direct relationship between usability and software architecture to be able to describe and categorize our usability patterns in such a way that they can be used as requirements that can support architectural design. ...  
 Cited by 39 - Related articles - BL Direct - All 12 versions - Import into RefWorks

[Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces](#)

J Lin... - Proceeding of the twenty-sixth annual SIGCHI ... 2008 - portal.acm.org  
 ... in SUEDE, would be more suitable for more extensive usability tests. ... The patterns are stored in external files in an XML-based format called the Pattern Language Markup Language ... Damask's architecture is based on the model-view-controller (MVC) software design pattern. ...

Cited by 42 · Related articles · BL Direct · All 7 versions · Import into RefWorks

**Design components: toward software composition at the design level**

RK Keller... · ... 20th international conference on **Software** ... , 1998 - portal.acm.org  
 ... The criteria that lead to a certain **design** include functionality, reliability, **usability**, efficiency, main-tainability, and portability [15]. ... In this pa-per, we introduced an approach in which **design patterns** constitute the foundation of **software** development. ...  
 Cited by 114 · Related articles · BL Direct · All 9 versions · Import into RefWorks

[PDF] from psu.edu

**Usability Patterns for Applications on the world wide web**

K Perzelski... · Proc. Semantic Web Enabled **Software** ... , 2005 - Citeseer  
 ... For example the concept of **design** to a graphic artist is very different than the concept of **design** to a **software** engineer. **Usability** is a critical success factor for successful web applications. **Patterns** can provide a language for communicating **usability** concerns among these ...  
 Cited by 33 · Related articles · View as HTML · All 9 versions · Import into RefWorks

[PDF] from hillside.net

**Human-centered software engineering**

A Seflah, J Guliksen... · ... 2005 - 2006.cusec.net  
 ... engineered? Peanut Butter Theory of **Usability** Training Resources [Seflah-93] User ... Interface Process Orientation Guidelines and **Patterns** [Seflah-01] Page 16. Moving to Human-Centered **Software** Development Human-centered development User-driven Solution focus ...  
 Cited by 29 · Related articles · View as HTML · All 5 versions · Import into RefWorks

[PDF] from cusec.net

**Large-scale C++ software design**

J Lakos · Reading, MA, 1996 - books.google.com  
 ... Page 144. 124 LARGE-SCALE C++ **SOFTWARE DESIGN** Figure 5. A large program | MAIN\* C] Figure 6. A large system. tern architecture. Reduced testability, reduced **usability**, and reduced modifi- bility are important primary causes. ...  
 Cited by 235 · Related articles · All 13 versions · Import into RefWorks

[PDF] from brown.edu

**Multiple user interfaces: Towards a task-driven and patterns-oriented design model**

A Seflah... · ... Systems, Design, Specification, and Verification, 2002 - Springer  
 ... **Patterns** should not be considered just as an alternative **design** tool to guide-lines ... For instance, CASE tools have long been available to assist **software** developers to integrate the ... in- tegrated with existing development tools in order to maximize the benefits of **usability patterns**. ...  
 Cited by 25 · Related articles · BL Direct · All 5 versions · Import into RefWorks

**Managing to meet usability requirements: establishing and meeting software development goals**

JL Bennett · Visual display terminals, 1984  
 Cited by 69 · Related articles · All 2 versions · Import into RefWorks

**Bridging patterns: An approach to bridge gaps between SE and HCI**

E Folmer, M Welie... · Information and **Software** Technology, 2006 - Elsevier  
 ... can assess what it means in their context and can decide whether they need to modify the **software** architecture to support these **patterns**. ... by adaptive maintenance activities once the system has been implemented and leads to architectures with better support for **usability**. ...  
 Cited by 46 · Related articles · All 6 versions · Import into RefWorks

[PDF] from rug.nl

**Capturing and disseminating usability patterns with semantic web technology**

S Henninger, M Keshk... · ... and Perspectives on HCI **Patterns**, 2003 - Citeseer  
 ... detail. First, given current **software** development and **usability** processes, efforts to apply **patterns** fall short of the goal to put the accumulated knowledge of user-centered **design** at the fingertips of **software** developers. Just as ...  
 Cited by 6 · Related articles · View as HTML · All 9 versions · Import into RefWorks

[PDF] from psu.edu

**Achieving usability through software architecture**

... · MELLON UNIV PITTSBURGH PA **SOFTWARE** ... · 2001 - Citeseer  
 ... The architecture **patterns** we provide will enable **usability** specialists to evaluate the impact of ... give these specialists the tools necessary to decide which aspects of **usability** should be ... We also hope to give **software** engineers the tools necessary to understand particular aspects ...  
 Cited by 81 · Related articles · View as HTML · BL Direct · All 25 versions · Import into RefWorks

[PDF] from psu.edu

**Finding a place for discount usability engineering in agile development: throwing down the gauntlet**

D Kane · 2003 - computer.org  
 ... This example illustrates the competing forces that drive user interface **design** decisions. Just as **design patterns** capture the tradeoffs of **software design** decisions, user interface **patterns** can help developers make better choices about tradeoffs that affect **usability**. ...  
 Cited by 17 · Related articles · All 10 versions · Import into RefWorks

[PDF] from agilealliance.

**Combining usability techniques to design geovisualization tools for epidemiology**

AC Robinson, J Chen, EJ Lengerich... · Cartography and ... , 2005 - ncbi.nlm.nih.gov  
 ... 1999;19(6):51-9. Haklay M, Tobon C. **Usability** evaluation and PPGIS: Towards a user ... **Patterns** of colorectal cancer incidence, risk factors, and screening in Kentucky. ... Jacquez GM, Estberg L. BioMedware, Inc.; Clusterser 2.0: **Software** for the detection and analysis of spatial ...  
 Cited by 39 · Related articles · All 15 versions · Import into RefWorks

[HTML] from nih.gov

**Usability and open source software**

DM Nichols, M Twidale... · 2002 - Citeseer  
 ... These approaches are not necessarily restricted to OSS; several can be applied to proprietary **software**. Indeed the ideas derived from discount **usability** engineering and participatory **design** originated in developing better proprietary **software**. ...  
 Cited by 146 · Related articles · View as HTML · All 38 versions · Import into RefWorks

[PDF] from psu.edu

**An Ontology-Based Infrastructure for Usability Design Patterns**

S Henninger... · Proc. Semantic Web Enabled **Software** ... , 2005 - Citeseer  
 Abstract. **Usability patterns** represent knowledge about known ways to **design** graphical user interfaces that are usable and meet the needs and expectations of users. There is currently a plethora of **usability patterns** published in books, private repositories and the World-Wide ...  
 Cited by 11 · Related articles · View as HTML · All 5 versions · Import into RefWorks

[PDF] from psu.edu

**Instantiating and detecting design patterns: Putting bits and pieces together**

H Albin-Amiot, P Cointe, YG Guéhéneuc... · ase, 2001 - computer.org  
 ... We are currently assessing the **usability** and suit- ability of our tools on several frameworks: Java ... pat- terns, to automate, or to assist, in designing, under- standing, and re-engineering **software**. ... we hope to see other approaches to this problem, based on the **patterns** we have ...  
 Cited by 88 · Related articles · All 20 versions · Import into RefWorks

[PDF] from psu.edu

**Multimedia effects on learning: Design implications of an integrated model**

T Hede... · Untangling the web: Establishing learning links. ... , 2002 - ascilite.org.au  
 ... to standardise **software** interface **design** as much as possible. Creating interfaces that conform to user expectations can reduce the overhead and learning demand of the **software** itself and allow learner focus on the material being presented. **Usability patterns** can provide the ...  
 Cited by 23 · Related articles · Cached · All 3 versions · Import into RefWorks

[HTML] from ascilite.org.

**Pattern languages in interaction design: Structure and organization**

M Van Welie... · Proceedings of interact, 2003 - welie.com  
 ... Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995). **Design Patterns**: Elements of Reusable Object- Oriented **Software**. Addison-Wesley, Reading, Mass. Graham, I. (2003). A pattern language for Web **Usability**. Addison-Wesley, Boston, US. ...  
 Cited by 107 · Related articles · View as HTML · All 8 versions · Import into RefWorks

[PDF] from welie.com

**About face: The essentials of user interface design**

A Cooper · 1995 - portal.acm.org  
 ... Applicable to multimedia and Web sites as well as application **software**, About Face is an invaluable resource for **design** professionals. top of page AUTHORS. Alan Cooper No contact information provided yet. Bibliometrics: publication history PUBLICATION years, 1995-2007. ...  
 Cited by 304 · Related articles · All 2 versions · Import into RefWorks

**Finding the pattern you need: The design pattern intent ontology**

H Kampffmeyer... · Model Driven Engineering Languages and ... , 2007 - Springer  
 ... This reduces the **usability** of ontologies to domain experts only. ... It allows **design** problems to be described visually and suggests a set of matching **design patterns** for a given **design** problem. ... It shows the applicability of ontologies for tool support in the area of **software design**. ...  
 Cited by 17 · Related articles · BL Direct · All 10 versions · Import into RefWorks

[PDF] from psu.edu

**The usability engineering life cycle**

J Nielsen · COMPUTER... , 1992 - computer.org  
 ... To ensure the **usability** of interactive computer products, we must actively include **usability** concerns in the **software** development process. Of course, nobody deliberately sets out to **design** an unusable interface, but only a systematic **usability** effort using established methods ...  
 Cited by 226 · Related articles · All 6 versions · Import into RefWorks

**Notes on a pattern language for interactive usability**

G Casaday · CHI'97 extended abstracts on Human factors in. ... , 1997 - portal.acm.org  
 ... 10. IEEE. Special Issue on Object Methods, **Patterns**, and Architectures. IEEE **Software**, (January/February 1996). ... University of Chicago Press, 1970. 12. Jakob Nielsen, **Usability** Engineering, Morgan Kaufmann Publishers Inc., San Francisco, CA, 1993. ...  
 Cited by 24 · Related articles · All 2 versions · Import into RefWorks

**Design reuse through frameworks and patterns**

PW Fach · IEEE **Software**, 2001 - computer.org  
 ... Thus, changes for pure **usability** reasons have little chance of implementation in the later ... **design** process of frameworks, help develop suitable metaphors for **design patterns**, and above ... for specially trained HCI consultants but also for highly experienced **software** architects who ...  
 Cited by 19 · Related articles · BL Direct · All 11 versions · Import into RefWorks

[PDF] from psu.edu

**Usability-supporting architectural patterns**

L Bass, BE John, N Juristo... · 2004 - computer.org  
 ... In this sense, our **usability**-supporting architectural **patterns** differ from other architectural **patterns** in that most other **patterns** are presented as if they ... Our long term goal is to develop a handbook of USAPs that has utility to both **software** developers and **usability** engineers. ...  
 Cited by 7 · Related articles · BL Direct · All 11 versions · Import into RefWorks

Web Images Videos Maps Books Translate Gmail more ▾ laura.carvajal@gmail.com | Scholar Preferences | My Account | Sign out

Google scholar architectural patterns usability Search Advanced Scholar Search

Scholar Articles excluding patents ▾ anytime ▾ include citations ▾ Create email alert Results 1 - 100 of

**Improving software usability through architectural patterns** [PDF] from psu.edu  
 N Juristo, M Lopez, AM Moreno... - Bridging the Gaps Between... 2003 - Citeseer  
 Improving software usability through architectural patterns Natalia Juristo School of Computing-Universidad Politécnica de Madrid, Spain Marta Lopez School of Computing-Universidad Complutense de Madrid, Spain Ana M. Moreno School of Computing-Universidad ...  
 Cited by 31 - Related articles - View as HTML - All 11 versions - Import into RefWorks

**Linking usability to software architecture patterns through general scenarios** [PDF] from psu.edu  
 L Bass... - Journal of Systems and Software, 2003 - Elsevier  
 ... Although many of the general scenarios presented here will be applicable to other paradigms, these environments are likely to introduce their own additional usability requirements. Understanding the effect on other attributes of usability architectural patterns. ...  
 Cited by 92 - Related articles - All 10 versions - Import into RefWorks

**Architectural patterns for enabling application security** [PDF] from psu.edu  
 J Yoder... - Urbana, 1998 - Citeseer  
 Page 1. Architectural Patterns for Enabling Application Security Joseph Yoder Department of Computer Science University of Illinois at Urbana-Champaign Urbana, IL 61801 jyoder@uiuc.edu Jeffrey Barcalow Reuters Information ...  
 Cited by 210 - Related articles - View as HTML - All 35 versions - Import into RefWorks

**Supporting usability through software architecture**  
 L Bass... - Computer, 2001 - ieeexplore.ieee.org  
 ... However, separation-based architectural patterns are independent of these mechanisms and are insufficient to provide the ability to support a cancel command. USABILITY FACETS At the SEI, we have isolated 26 usability facets that require software architectural ...  
 Cited by 36 - Related articles - BL Direct - All 8 versions - Import into RefWorks

**Architecting for usability: a survey** [PDF] from psu.edu  
 E Folmer... - Journal of systems and software, 2004 - Elsevier  
 ... often requires the use of certain design patterns or styles. For instance, to improve portability and modifiability it may be beneficial to use a layered architecture style. It is our conjecture that a large number of issues associated to usability may also require architectural support in ...  
 Cited by 138 - Related articles - All 12 versions - Import into RefWorks

**Architectural patterns revisited—A pattern language** [PDF] from univie.ac  
 P Avgeriou, U Zdun - 2005 - eprints.cs.univie.ac.at  
 Page 1. Architectural Patterns Revisited – A Pattern Language Paris Avgeriou Uwe Zdun CONCERT ... Regrettably, finding and applying the appropriate architectural patterns in practice still remains largely ad-hoc and unsystematic. This ...  
 Cited by 74 - Related articles - View as HTML - All 4 versions - Import into RefWorks

**Quality-Attribute Based Economic Valuation of Architectural Patterns** [PDF] from psu.edu  
 I Ozkaya, R Kazman... - on The Economics of Software and ..., 2007 - portal.acm.org  
 ... Many architectural decisions regarding usability are captured at the architectural level as a series of modularization decisions, such as the use of the model-view-controller pattern. However, many architecturally significant usability decisions require other patterns [14] as well ...  
 Cited by 17 - Related articles - All 18 versions - Import into RefWorks

**Usability-supporting architectural patterns**  
 L Bass, BE John, N Juristo... - 2004 - computer.org  
 Page 1. Usability-supporting Architectural Patterns 1 Len Bass Carnegie ... application. In this tutorial, we present usability-supporting architectural patterns. Each pattern describes a usability concern that is not supported by separation alone. ...  
 Cited by 7 - Related articles - BL Direct - All 11 versions - Import into RefWorks

**Object-oriented software engineering** [PDF] from sut.ac.th  
 T Lethbridge... - 2001 - sutlib2.sut.ac.th  
 ... centered design 254 7.2 Characteristics of users 256 7.3 The basics of user interface design 258 7.4 Usability principles 262 ... Techniques for making good design decisions 336 9.4 Model Driven Development 340 9.5 Software architecture 342 9.6 Architectural patterns 347 The ...  
 Cited by 122 - Related articles - All 5 versions - Import into RefWorks

**A software architectural view of usability patterns** [PDF] from psu.edu  
 X Ferré, N Juristo, AM Moreno... - Proceedings of INTERACT... 2003 - Citeseer  
 ... 3 Architectural Usability Patterns The most widely used concept of pattern in software development is the design pattern, and it is used particularly in the object-oriented paradigm. ...  
 Page 5. 3.1 Procedure for outputting architectural patterns for usability ...

Cited by 9 - Related articles - View as HTML - All 9 versions - Import into RefWorks

**Ajax design patterns**  
 M Mahemoff - 2006 - portal.acm.org  
 ... Functionality and usability: Describes the types of user interfaces you'll come across in Ajax applications, as well as the new types of functionality that Ajax makes possible. ...  
 The book also covers some architectural patterns too. ...  
 Cited by 88 - Related articles - All 5 versions - Import into RefWorks

**Bringing usability concerns to the design of software architecture** [PDF] from psu.edu  
 BE John, L Bass, MI Sanchez-Segura... - Computer Interaction and ..., 2005 - Springer  
 ... In this paper, we introduce usability-supporting architectural patterns. A specific solution in this form does not provide good guidance for architects who will come to the usability-supporting architectural patterns after having made a number of overarching design decisions. ...  
 Cited by 28 - Related articles - BL Direct - All 13 versions - Import into RefWorks

**Software architecture analysis of usability** [PDF] from psu.edu  
 E Folmer, J van Gurp... - Engineering Human Computer ..., 2005 - Springer  
 ... Actions for multiple objects may be implemented by the composite pattern [9] or the visitor pattern [9]. (Positive) relationships have been defined between the elements of the framework that link architectural sensitive usability patterns to usability properties and attributes. ...  
 Cited by 34 - Related articles - BL Direct - All 27 versions - Import into RefWorks

**A framework for capturing the relationship between usability and software architecture** [PDF] from psu.edu  
 E Folmer, J Van Gurp... - Process: Improvement and ..., 2003 - Wiley Online Library  
 ... Fixing usability issues during this late stage of the development proves to be very costly. Some usability-improving modifications such as usability patterns may have architectural implications. We believe that the software architecture may restrict usability. ...  
 Cited by 39 - Related articles - BL Direct - All 12 versions - Import into RefWorks

**Pattern-oriented software architecture: a system of patterns** [PDF] from ispras.ru  
 F Buschmann, R Meunier, H Rohrer, P Sommerlad... - 2008 - Wiley-India  
 Cited by 4090 - Related articles - All 13 versions - Import into RefWorks

**Using patterns to capture architectural decisions** [HTML] from univie.ac  
 NB Harrison, P Avgeriou... - IEEE software, 2007 - doi.ieeecomputersociety.org  
 ... Using patterns: Practical considerations. Back to Top. Architectural design is an especially challenging decision-making process because it involves frequent ... For example, deciding to implement a certain security approach might impact the system's performance and usability. ...  
 Cited by 33 - Related articles - BL Direct - All 11 versions - Import into RefWorks

**Architectural patterns for collaborative applications** [PDF] from rug.nl  
 P Avgeriou... - journal of computer applications in technology, 2006 - Inderscience  
 ... Architectural patterns for collaborative applications ... This paper seeks to provide design reuse in the form of architectural patterns that focus on low-level horizontal issues: distribution, message exchange, functional decomposition, sharing data, concurrency and synchronisation. ...  
 Cited by 19 - Related articles - BL Direct - All 11 versions - Import into RefWorks

**Preparing Usability Supporting Architectural Patterns for Industrial Use** [PDF] from psu.edu  
 P Stoll, BE John, L Bass... - Workshop on Interplay between Usability - Citeseer  
 Preparing Usability Supporting Architectural Patterns for Industrial Use Pia Stoll Bonnie E. John, Len Bass, Elspeth Golden ABB Corporate Research Forskargränd 6 SE 72178 Västerås, Sweden Tel: +46 21 32 30 00 Carnegie Mellon University 5000 Forbes Ave. ...  
 Cited by 4 - Related articles - View as HTML - All 14 versions - Import into RefWorks

**A method to elicit architecturally sensitive usability requirements: its integration into a software development process**  
 T Rafia, PN Robillard... - Software Quality Journal, 2007 - Springer  
 ... Bass and John (2003) and Folmer et al. (2003a) are among the early researchers who investigated these issues and spelled out the potential links that remain between usability requirements and architectural patterns. ... 2.1 Usability-driven software architectural patterns ...  
 Cited by 19 - Related articles - BL Direct - All 4 versions - Import into RefWorks

**Usability patterns in software architecture** [PDF] from psu.edu  
 E Folmer... - Human-computer interaction: theory and ..., 2003 - books.google.com  
 ... system. Our research has identified several usability patterns that require architectural support. We ... framework. Future research should focus on verifying the architectural sensitiveness of the usability patterns that have been identified. For ...  
 Cited by 15 - Related articles - All 10 versions - Import into RefWorks

**Bridging patterns: An approach to bridge gaps between SE and HCI** [PDF] from rug.nl  
 E Folmer, M Welie... - Information and Software Technology, 2006 - Elsevier  
 ... for UI patterns. Bass et al. [4] identified scenarios that illustrate particular aspects of usability that are architecture-sensitive and suggest architectural patterns for implementing these scenarios. A framework, which expresses ...

Cited by 46 - Related articles - All 6 versions - Import into RefWorks

**Scenario-based assessment of software architecture usability**

E Folmer, J Van Gurp... - Bridging the Gaps Between Software ... 2003 - Citeseer  
 ... The framework is used for extracting information regarding the **architectural** information related to **usability** required for the assessment. The framework consists of the following concepts: • **Usability** attributes. • **Usability** properties. • **Usability** patterns. ...  
 Cited by 22 - Related articles - View as HTML - All 18 versions - Import into RefWorks

[PDF] from psu.edu

**The value of a usability-supporting architectural pattern in software architecture design: a controlled experiment**

E Golden, BE John... - ... of the 27th international conference on ... 2005 - portal.acm.org  
 ... ABSTRACT Design **patterns** have been claimed to facilitate modification and improve understanding in software design. A controlled experiment was performed to assess the usefulness of portions of a **Usability**-Supporting **Architectural** Pattern (USAP) in modifying the ...  
 Cited by 25 - Related articles - BL Direct - All 11 versions - Import into RefWorks

[PDF] from psu.edu

**Sonification design patterns**

S Barrass - Proceedings of the 2003 International Conference on ... - icad.org  
 ... Although Design **Patterns** were originally **architectural**, **Patterns** have gained prominence as a method for designing computer software [Gamma ... of Programming (LoP) holds regular workshops on writing **Patterns** and there are now **Patterns** for **Usability**, Interaction Design ...  
 Cited by 26 - Related articles - View as HTML - All 4 versions - Import into RefWorks

[PDF] from icad.org

**Building application frameworks: object-oriented foundations of framework design**

M Fayad, DC Schmidt... - 1999 - John Wiley & Sons Inc  
 Cited by 455 - Related articles - All 2 versions - Import into RefWorks

**UPI: a software development process aiming at usability, productivity and integration**

K Sousa, E Furtado... - Proceedings of the 2005 Latin ... 2005 - portal.acm.org  
 ... including **usability** patterns in it. They propose an extension of the Model-View-Controller (MVC) design pattern (MVC, 2000) by defining four modules to specify **usability** architectural patterns. These modules are: active modules ...  
 Cited by 16 - Related articles - All 7 versions - Import into RefWorks

[PDF] from ucl.ac.be

**A responsibility-based pattern language for usability-supporting architectural patterns**

BE John, E Bass, E Golden... - Proceedings of the 1st ACM ... 2009 - portal.acm.org  
 Bonnie E. John HCI Institute Carnegie Mellon University 5000 Forbes Ave, Pittsburgh, PA, 15217 +1-412-268-7182 bej@cs.cmu.edu ... Len Bass Software Engineering Institute Carnegie Mellon University 4500 Fifth Ave, Pittsburgh, PA, 15217 +1-412-268-6763 ljb@sei.cmu.edu  
 Cited by 3 - Related articles - All 6 versions - Import into RefWorks

[PDF] from diva-porte

**Leveraging architecture patterns to satisfy quality attributes**

N Harrison... - Software Architecture, 2007 - Springer  
 ... used the ISO quality model [14], which contains functionality, reliability, **usability**, efficiency, maintainability ... While the book gives several variants of the **patterns**, we limited this analysis ... pattern, it was designated as "key." This differentiation supports **architectural** reasoning: used ...  
 Cited by 13 - Related articles - BL Direct - All 9 versions - Import into RefWorks

[PDF] from rug.nl

**Modeling the variability of architectural patterns**

AW Kamal... - Proceedings of the 2010 ACM Symposium ... 2010 - portal.acm.org  
 ... specialized pattern participating. We believe that we can cover more **architectural** patterns in the near future, which will provide a better re-**usability** support to the architects for systematically expressing **architectural** patterns variants. ...  
 Cited by 8 - Related articles - All 3 versions - Import into RefWorks

[PDF] from rug.nl

**Architectural styles and the design of network-based software architectures**

RT Fielding - 2000 - Citeseer  
 Page 1. UNIVERSITY OF CALIFORNIA, IRVINE **Architectural** Styles and the Design of Network-based Software Architectures DISSERTATION ... The Web's **architectural** style was developed iteratively over a six year period, but primarily during the first six months of 1995. ...  
 Cited by 1699 - Related articles - View as HTML - All 76 versions - Import into RefWorks

[PDF] from psu.edu

**Interaction design patterns: twelve theses**

JO Borchers - Workshop, The Hague, 2000 - Citeseer  
 ... the application do- main to enhance communication in interdisciplinary design teams, and outline how those pattern languages fit into the **usability** engineering lifecycle. ... The medium that **architectural** patterns use to sensitize the reader to the subject of a pattern is a photograph. ...  
 Cited by 17 - Related articles - View as HTML - All 17 versions - Import into RefWorks

[PDF] from psu.edu

**From usability tasks to usable user interfaces**

K Sousa... - Proceedings of the 4th international workshop ... 2005 - portal.acm.org  
 ... 2004. 6. Juristo, N.; Lopez, M.; Moreno, A.; Sánchez, M. Improving Software **Usability** through **Architectural** Patterns. In: Workshop Bridging the Gaps between SE and HCI - International Conference on Software Engineering (ICSE), 2003, USA, 2003, pp. 12-19. ...

[PDF] from ucl.ac.be

Cited by 11 - Related articles - All 7 versions - Import into RefWorks

**Modeling architectural patterns' behavior using architectural primitives**

A Kamal... - Software Architecture, 2008 - Springer  
 ... We believe that in different **architectural** views, more primitives will be discovered in the near future, which will provide a better re-**usability** support to the architects for systematically expressing **architectural** patterns. References ...  
 Cited by 7 - Related articles - All 5 versions - Import into RefWorks

[PDF] from rug.nl

**Scenarios, quality attributes, and patterns: Capturing and using their synergistic relationships for product line architectures**

MA Babar - 2004 - computer.org  
 ... Software architecture (SA) of a product family constrains the achievement of various quality attributes (such as reusability, performance, security, maintainability and **usability**) [1]. A number of ... These approaches heavily depend on **architectural** styles and **patterns** to design ...  
 Cited by 9 - Related articles - All 5 versions - Import into RefWorks

**A cookbook for using the model-view controller user interface paradigm in Smalltalk-80**

GE Krasner... - Journal of Object-oriented programming, 1988 - portal.acm.org  
 ... Atsuto Kubo, Hironori Washizaki, Yoshiaki Fukazawa, A metric for measuring the abstraction level of design **patterns**, Proceedings of the ... Leszek A. Maciaszek, "Roundtrip **architectural** modeling", Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling, p.17 ...  
 Cited by 1418 - Related articles - All 4 versions - Import into RefWorks

**Reconciling usability and interactive system architecture using patterns**

A Sefah, T Mohamed, H Habieb-Mammar... - Journal of Systems and ... 2008 - Elsevier  
 ... 4. Illustrate, as part of the pattern documentation, how these **patterns** can be applied within existing **architectural** models such as MVC. 3. Identifying and categorizing typical scenarios. The first step in our approach for achieving **usability** via software architecture and **patterns** is to ...  
 Cited by 12 - Related articles - All 2 versions - Import into RefWorks

**Designing component-based frameworks using patterns in the UML**

G Larsen - Communications of the ACM, 1999 - portal.acm.org  
 ... **Architectural** patterns or frameworks provide descriptions of the software architecture. ... for the engineers who will be using your framework to insure consistency, quality, and **usability**. ... the process of using the UML to design a component-based framework using existing **patterns**. ...  
 Cited by 56 - Related articles - BL Direct - All 3 versions - Import into RefWorks

**Mining patterns to support software architecture evaluation**

L Zhu, MA Babar... - 2004 - computer.org  
 ... are an important source of quality attribute sensitive scenarios and other **architectural** information. ... a number of pieces of architecturally vital information from well know software **patterns**. ... requirement of a software system, eg, reliability, modifiability, performance, **usability** and so ...  
 Cited by 26 - Related articles - All 9 versions - Import into RefWorks

[PDF] from psu.edu

**Helping software architects design for usability**

E Golden - Proceedings of the 1st ACM SIGCHI symposium on ... 2009 - portal.acm.org  
 ... 13. Juristo, N, Lopez, M, Moreno, A, and Sanchez-Segura, M. 4. Improving software **usability** through **architectural** patterns. in ICSE 2003 Workshop on Bridging the Gaps Between Software Engineering and Human-Computer Interaction (Portland, OR, 2003), 14. ...  
 Cited by 6 - Related articles - Import into RefWorks

**Improving Web information systems with navigational patterns**

G Rossi, D Schwabe... - Computer Networks, 1999 - Elsevier  
 ... They are part of a catalogue of around twenty **architectural**, navigational and user interface **patterns** (see [3, 11, 12]). ... 2.1.4. Related **patterns**. ... The latter approach is clearly not desirable because the site's **usability** is greatly reduced and it may become unmanageable as it grows. ...  
 Cited by 68 - Related articles - All 8 versions - Import into RefWorks

[HTML] from www8.o

**A goal-based organizational perspective on multi-agent architectures**

M Kolp, P Giorgini... - Intelligent Agents VIII, 2002 - Springer  
 ... Integrity **Usability** Order Processor Confidentiality ... The alliance revolution : the new shape of business rivalry, Harvard University Press, 1996. [10] S. Hayden, C. Carrick, and O. Yang. "Architectural Design Patterns for Multiagent Coordination". In Proc. of the 3rd Int. ...  
 Cited by 115 - Related articles - BL Direct - All 11 versions - Import into RefWorks

**Quality attribute design primitives and the attribute driven design method**

L Bass, M Klein... - Software Product-Family Engineering, 2002 - Springer  
 ... The software architecture community generally believes that quality attributes (such as performance, **usability**, security, reliability and modifiability) of a ... We have embarked on an effort to identify and codify **architectural** patterns that are primitive with respect to the achievement of ...  
 Cited by 97 - Related articles - BL Direct - All 17 versions - Import into RefWorks

[PDF] from psu.edu

**Preserving software quality characteristics from requirements analysis to architectural design**

H Schmidt... - Software Architecture, 2006 - Springer

[PDF] from uni-due.d



## 9.4 System Requirement Specifications

The following sections present the SRS documents for the three projects used in the validation of the proposed solution. These are the original SRS document, written in Spanish, the main language spoken by the selected test subjects.

### 9.4.1 Online task manager (Gestor de Tareas Online)

El Gestor de Tareas Online es un sistema web que permite al usuario manejar listas de tareas pendientes de manera interactiva. Provee al usuario la capacidad de organizar tareas en múltiples listas, así como planificarlas en el tiempo y visualizarlas bajo múltiples perspectivas. Mantiene al usuario informado sobre el estado de sus tareas y permite que sean compartidas con otros usuarios del sistema. Este sistema se basa parcialmente en los servicios provistos por *Remember the Milk*.

#### 9.4.1.1 Software Requirements

**(R-01)** Permitir al usuario autenticarse (*login*) introduciendo su *username* y *password*. Si el proceso de autenticación tarda más de dos segundos, el sistema deberá mostrar al usuario un spinning wheel y un mensaje temporal que diga “autenticando, por favor espere”

**(R-02)** Permitir al usuario salir del sistema (*logout*) en un único paso, i.e. haciendo clic en un link de salida. Si el proceso de salir del sistema tarda más de dos segundos, el sistema deberá mostrar al usuario un spinning wheel y un mensaje temporal que diga “saliendo, por favor espere”

**(R-03)** Permitir la creación de tareas. Para crear una tarea, el usuario debe seleccionar primero la Lista de Tareas (“Lista Principal” estará seleccionada por defecto) a la cual desea agregar la tarea. Deberá seleccionar la Lista de un combo-box que le ofrece el sistema para este fin, el cual contiene los nombres de todas las Listas de Tareas existentes en el sistema para este usuario. Luego de seleccionada una Lista de Tareas, el usuario elige la opción de “crear tarea” e introduce el nombre de la tarea a crear en un único campo de texto que le presenta el sistema. Al hacer clic en la tecla enter, la tarea se crea y se presentan al usuario opciones de modificación especificadas en (R-04). Cada tarea creada aparecerá automáticamente en el *combo-box* la próxima vez que éste sea consultado.

**(R-04)** Al hacer clic sobre una tarea de cualquier lista o justo después de la creación de una nueva tarea, el sistema debe mostrar al usuario opciones de modificación. Estas opciones se mostrarán dentro de la misma ventana, en el espacio de la pantalla reservado para tal fin. Para cada tarea se podrá modificar:

- su fecha límite, que representa la fecha para la cual deberá haberse completado la tarea. Se presentarán tres campos de dos dígitos cada uno en los que el usuario introducirá el día, el mes y el año. El valor por defecto de una tarea nueva o de una que nunca ha sido modificada será “0” en las tres casillas, lo que indica que la misma no tiene fecha límite.
- el tiempo estimado para completarla. Los valores a introducir deberán tener venir expresados en número de días. Se presentará al usuario un campo de tres dígitos seguido de la palabra “días”. El valor por defecto de una tarea nueva o de una que nunca ha sido modificada será “0 días”, lo que indica que la misma no tiene tiempo estimado.

Una vez editada la información necesaria debe guardar los cambios, para lo que el usuario hará clic en el botón de “Guardar”. Inmediatamente después, el botón deberá permanecer deshabilitado hasta que se haya guardado la tarea completamente. Una vez guardada la tarea, el botón debe volver a su estado normal.

Simultáneamente, y solo si el proceso de guardar dura más de 2 segundos, se mostrará un spinning wheel para indicar el progreso hasta que se haya guardado la tarea.

La acción de “guardar” deberá poderse deshacer. Una vez ejecutada la opción de guardar se debe mostrar un mensaje al usuario en la parte superior de la aplicación que le pregunte si desea deshacer esta última operación. Este mensaje debe desaparecer después de 10 segundos.

**(R-05)** Para borrar una (o varias) tarea(s), el usuario podrá elegirla(s) de la lista de tareas correspondiente y elegir la opción de borrar. Esta acción deberá poderse deshacer. Una vez ejecutada la opción de borrar se muestra un mensaje al usuario en la parte superior de la aplicación que le pregunte si desea deshacer esta última operación. Este mensaje debe desaparecer después de 10 segundos.

**(R-06)** El sistema debe permitir al usuario la opción de trabajar en modo offline, lo que implica que se desactiven todas las opciones de comunicación con otros sistemas y usuarios. Para este fin, el usuario elegirá la opción “Go offline” y el sistema quedará desconectado, mostrando en el área de estado de la aplicación (no del navegador) el símbolo de offline (i.e. círculo rojo) en lugar del de online (i.e. círculo verde) que se muestra en cualquier otro caso. Siempre que el usuario esté offline, la opción de “Go Offline” desaparecerá y aparecerá la opción de “Go Online” en su lugar. Para volver al modo online, el usuario hará clic en “Go online” y se habilitarán nuevamente las opciones de comunicación y el área de estado de la aplicación mostrará nuevamente el símbolo de online.

**(R-07)** El sistema debe permitir al usuario la opción de exportar su(s) lista(s) a un archivo de texto. Para esto elegirá la(s) lista(s) que desea exportar del *combo-box* correspondiente y hará clic en “exportar”. Si este proceso dura más de 10 segundos, se deberá mostrar una barra de progreso que indique el porcentaje de información real exportada de manera continua hasta que culmine el proceso. Se deberá también proveer la opción de cancelar el proceso, lo cual resultaría en datos parcialmente exportados.

**(R-08)** Permitir al usuario registrarse por primera vez. Le presentará un formulario en varios pasos. El primer paso le pedirá sus datos personales: nombre, apellido, email, país, ciudad. El segundo y último paso le pedirá que introduzca un nombre de usuario, una contraseña, la confirmación de la contraseña y una pregunta y respuesta de seguridad.

Durante este proceso el usuario deberá poder ir al paso anterior/siguiente mediante los botones “<” y “>” respectivamente, sin perder la información ya introducida. También podrá navegar entre pasos mediante los breadcrumbs que deberán estar siempre visibles durante el proceso de registro (cada elemento del breadcrumbs será un link al paso correspondiente).

**(R-09)** Cada vez que el usuario coloque el cursor sobre una tarea de cualquier lista, el sistema deberá mostrarle un tooltip con la fecha de culminación de la tarea.

**(R-10)** El sistema debe permitir al usuario crear nuevas Listas de Tareas. Para ello introduce el nombre de la tarea a crear en un único campo de texto que le presenta el sistema. Al hacer clic en la tecla “enter”, la Lista de Tareas (vacía) se crea y está disponible para ser usada como se especifica en (R-03) y en (R-10)

**(R-11)** El sistema debe permitir al usuario eliminar una Lista de Tareas. Para ello, elige la Lista que desea eliminar del *combo-box* de Listas de Tareas y selecciona la opción de “Eliminar”. Al hacerlo, el sistema muestra al usuario una advertencia, preguntándole si realmente quiere eliminar esta Lista de Tareas, advirtiéndole que al hacerlo estará eliminando todas las tareas contenidas en ella. La advertencia tiene dos opciones: OK y Cancelar. OK permite que continúe la eliminación de la tarea y Cancelar la ignora.

**(R-12)** El sistema debe presentar al usuario una sección donde pueda definir sus preferencias. Estas pueden ser las siguientes:

- Número máximo de tareas a mostrar por página para cada lista (por defecto serán 10)
- Número máximo de tareas a mostrar por cada lista en el Mapa de Listas (R-13). (Valor por defecto: 3. Valor máximo permitido: 10)
- Lista de Tareas que debe mostrar el sistema justo después de que el usuario se autentifique (por defecto será la Lista Principal)

Una vez establecidas las preferencias el sistema deberá permitir al usuario guardarlas. También permitirá restablecer los valores por defecto en cualquier momento.

**(R-13)** El sistema debe permitir al usuario ver sus Listas de Tareas de manera gráfica. Al entrar en la sección “Mapa de Listas” por primera vez el sistema muestra al usuario un recuadro que contiene todas sus listas. Cada lista esta representada por un contenedor rectangular que muestra el título de la lista y sus elementos (tareas) según las preferencias especificadas en (R-12). El usuario debe poder colocar las listas como desee (por columnas) dentro del recuadro principal así como crear nuevos mapas. Solo habrá un mapa activo en cada momento, el resto se podrá acceder mediante pestañas, cuyas listas se cargaran solamente cuando el mapa este activo. Para mover una lista a otro mapa, el usuario hará clic en el botón derecho y elegirá el mapa al que desea moverla de un menú flotante que mostrará el sistema. Este menú dinámico muestra los nombres de todos los mapas existentes en el sistema en ese momento. El anexo 1 muestra un ejemplo gráfico de este requisito.

**(R-14)** El sistema debe permitir al usuario crear nuevos mapas. Para ello hace clic en el botón Crear Mapa y e introduce el nombre deseado en un único campo de texto que le presenta el sistema. Al hacer clic en la tecla “enter”, aparece el mapa como una pestaña adicional como se muestra en el anexo 1.

**(R-15)** Para eliminar un mapa, el usuario hará clic en la pestaña correspondiente y elegirá la opción *eliminar* del menú flotante que le muestra el sistema. Al eliminar un mapa, las listas que contiene no se eliminan, sino pasan al mapa inmediatamente siguiente.

**(R-16)** El sistema debe permitir al usuario enviar un conjunto de tareas a otro usuario. Esta funcionalidad de comunicación--no disponible en modo offline, como se especifica en el requisito (R-06)—permite que el usuario seleccione una (o varias) tareas de cualquier lista y mediante un clic en el botón enviar el sistema le pedirá que introduzca el username del usuario al cual desea enviar estas tareas. Si el usuario destino se encuentra en modo online, le aparecerá una ventana de diálogo preguntándole si desea aceptar las tareas enviadas, incluyendo el nombre del usuario que las envía. En caso de aceptar, las tareas se agregan a la Lista Principal, de lo contrario, las tareas son rechazadas. El usuario que envía las tareas recibe una notificación en su correo electrónico indicándole el resultado de la operación.

**(R-17)** Permitir al usuario marcar cualquier tarea como “favorita” lo que la incluirá en una lista especial dinámica llamada Favoritos. A esta lista no se podrán agregar manualmente tareas, y mostrará únicamente aquellas que hayan sido marcadas como favoritas en otras listas.

#### 9.4.2 Home automation system (sistema de domótica del hogar)

Este proyecto consiste en el desarrollo del “front-end” de un sistema de control de domótica que permita la configuración y monitorización remota del comportamiento de la red de sensores y actuadores de un hogar. Deberá alertar al usuario en tiempo real sobre cualquier alteración relevante de sensores determinados y permitirle manipularlos de manera individual o grupal. Debido a la sensibilidad del tema de la domótica (que incluye, entre otras cosas,



activación de alarmas y cerraduras), es de suma importancia que el sistema resultante posea ciertas características de usabilidad cruciales para garantizar su uso adecuado. Estas características de usabilidad se detallan en la ERS inicial que se entregará al estudiante.

La aplicación podrá ser de escritorio o web, con la única restricción de que la comunicación con el con el “back-end” sea a través de Internet para permitir su control de manera remota.

El “back-end” es el conjunto de controladores que manejan el funcionamiento de los sensores y actuadores localmente. Su desarrollo es fuera del alcance de este proyecto, pero se requerirá la creación de un “stub” de software que simule su comunicación con el “front-end”.

#### 9.4.2.1 Front-End

El front-end es la consola de control en sí y puede ser desarrollado para web o escritorio. Procesa todas las órdenes enviadas por el usuario y las envía al back-end, que es la parte del sistema que se encargaría de ejecutarlas físicamente.

El front-end es un sistema completo e independiente, posee su propia base de datos local (donde almacena los datos de la vivienda, los usuarios, etc) y todo su procesamiento ocurre de manera local.

Las funcionalidades iniciales que deberá proveer al usuario son las siguientes:

##### 9.4.2.1.1 Requisitos Control de Actuadores

Req(1). Encender y apagar todas las luces de la vivienda de manera individual. Tanto el encendido como el apagado deberán poderse deshacer mediante una opción global de *Undo*, como ctrl-z.

Req(2). Encender y apagar el aire acondicionado, así como controlar su temperatura, la cual puede oscilar entre los 16 y los 30 grados. Igual que en el requisito 1, tanto el encendido como el apagado deberán poderse deshacer .

Req(3). Controlar persianas verticales fijas, cuyos paneles pueden girar de 15 a 165 grados. La posición con mayor paso de luz son los 90 grados.

Durante la ejecución de este proceso, que dura aproximadamente 1/10 seg por cada grado que se mueven las persianas, se deberá mostrar una barra de progreso indicando el porcentaje que se ha ejecutado de la acción.

Si durante el proceso el usuario elige la opción de cancelar, las persianas deberán cesar su movimiento.

Una vez terminada (o cancelada) la acción el usuario deberá poder deshacerla, lo que devolvería las persianas a su posición anterior .

Req(4). Abrir y cerrar la puerta del garaje. Este proceso tarda unos 35 segundos en completarse. Se deberá mostrar al usuario un indicador de progreso indeterminado (i.e. spinning wheel, reloj, etc)

Se proveerá también la opción de cancelar y deshacer. Ambas opciones devolverán a la puerta a su estado inmediato anterior a la ejecución.

Req(5). Activar la máquina de café. La máquina solo dispensará café si hay una taza colocada correctamente. De lo contrario se mostrará una notificación al usuario indicando que no se ha podido realizar la acción. Este proceso tarda unos 40 segundos y tiene varias etapas secuenciales: “calentando agua”(10s), “moliendo café” (5s), “colando”(25s) y “listo”.

Para indicar el progreso de ejecución se mostrará al usuario una lista de actividades y un “check” junto a cada actividad a medida que es completada .

Una vez culminada la ejecución se producirá una señal sonora para avisar al usuario que el café está listo.

#### 9.4.2.1.2 Macros

- Req(6). Se deberá permitir al usuario crear secuencias de acciones para ser ejecutadas posteriormente. Por ejemplo, si el usuario desea una función para apagar todas las luces, cerrar las persianas y activar la alarma (en lugar de ejecutar cada acción por separado), deberá permitírsele crear una “macro acción” que ejecute cada actividad en el orden solicitado, con un solo clic.
- Req(7). El usuario también deberá poder “grabar” secuencias de acciones. Al activar la opción de “grabar”, todas sus acciones subsiguientes quedarán registradas en una macro acción. Al elegir la opción de “detener” culminará la grabación.
- Req(8). En ambos casos (macros creadas y macros grabadas) se deberá permitir al usuario darle un nombre a la macro y la posibilidad de poderla invocar posteriormente tantas veces como desee.
- Req(9). Luego de ejecutado un macro, se debe permitir al usuario deshacer las acciones realizadas, una a una (exceptuando aquellas que no provean opción de *undo*).

#### 9.4.2.1.3 Alarmas y conexión con policía local

- Req(10). Se deberá proveer al usuario una opción de activar y desactivar la alarma principal de la vivienda. Cuando la alarma está activada deberá mostrarse en el área de estado un ícono indicativo de “activada” y un ícono opuesto cuando esté desactivada.
- Req(11). Si la alarma está activada y se detecta algún movimiento en la vivienda el status anterior pasará a un tercer estado llamado “en alerta” para el cual se deberá mostrar un tercer ícono que indique el estado de emergencia, además se mostrará un mensaje al usuario en el centro de la pantalla que le indique el estado de alerta.
- Req(12). Se proveerá la opción de conectar y desconectar la comunicación con la policía. Este estatus (conectado y desconectado) también deberá mostrarse con íconos autoexplicativos en el área de estado.  

El proceso de conectar con la policía tarda entre 10 y 40 segundos y deberá mostrarse al usuario el progreso (marcando-conectando-conectado) de la misma manera como se hizo en el requisito Req(5).
- Req(13). Tanto para activar y desactivar la alarma como para conectar y desconectar la comunicación con la policía el sistema mostrará al usuario una alerta, pidiéndole que introduzca sus credenciales para continuar. En el caso de que no las introduzca (i.e. que haga clic en el botón de cancel) o que introduzca credenciales erróneas, la orden quedará sin efecto y el sistema permanecerá en su estado actual.
- Req(14). El usuario deberá poder elegir el modo “ahorro de energía”, el cual apagará aquellas luces cuyas estancias no presenten movimiento durante un minuto. Si este modo está activado, deberá mostrarse un ícono indicativo en el área de estado, de lo contrario no se mostrará ningún ícono en relación a este estatus.
- Req(15). Si la alarma está conectada y el usuario intenta ejecutar alguna de las acciones listadas en los requisitos 3 a 5, el sistema deberá mostrar una alerta (con botones OK para seguir y Cancel para abortar) avisándole que la ejecución de estas acciones podría activar la alarma.

#### 9.4.2.1.4 Mapa Vivienda

**Req(16).** El plano de la vivienda debe poderse representar de manera gráfica, permitiendo al usuario seleccionar elementos como luces, persianas, etc. y ejecutar acciones sobre ellos. Cuando el usuario selecciona un elemento (por ejemplo, un punto de luz) el elemento deberá cambiar de color o enmarcarse en un contorno para dar seguridad al usuario de que ha seleccionado correctamente.

Una vez seleccionado un objeto se presentará al usuario un pequeño menú que liste las acciones que pueden ejecutarse sobre dicho objeto. El usuario debe poder elegir cualquiera de las acciones presentadas y ejecutarla con un clic. En dicho menú también debe presentarse la opción de Undo, si aplica, la cual desharía la última acción ejecutada sobre este objeto en particular.

**Req(17).** Cada vez que el usuario coloque el cursor sobre algún elemento de la vivienda sin hacer clic, deberá mostrarse un pop-up indicándole el nombre del elemento y su estado, por ejemplo “Luz 3 salon: apagada”

#### 9.4.2.2 Back-end

El back-end es la parte del sistema encargada de controlar físicamente los actuadores y sensores de una vivienda. Para este proyecto, el back-end se simulará mediante un pequeño módulo o “stub” que se comunicará con el front-end. Su implementación es libre pero se requerirá que los mensajes con el front-end se envíen y reciban utilizando algún protocolo independiente de la misma, como SOAP o similares. El back-end no requerirá UI y deberá desempeñar las siguientes funciones:

**Req(18).** Enviar algún tipo de acknowledge simple al front-end cada vez que éste le ordene ejecutar alguna acción sobre cualquiera de los actuadores (al encender una luz, al terminar de cerrar la puerta del garaje, etc), para que el front-end tenga la seguridad de que la operación que ha ordenado se ha realizado con éxito.

**Req(19).** Para los requisitos Req(3) a Req(5) y Req(12), simular también los tiempos de respuesta y/o los mensajes intermedios durante la ejecución:

Por ejemplo, en el caso del Req(5), cuando el front-end envíe la orden de iniciar la acción de preparado del café, el back-end deberá enviarle un primer mensaje de “calentando agua”, luego esperará 10s (el tiempo pautado para esta actividad) y enviará un segundo mensaje de “moliendo café”, y así hasta completar la actividad en el tiempo pautado. Al finalizar, enviará un mensaje de acknowledge como se indica en el Req(18)

**Req(20).** Para el Req(5), simular las respuestas del sensor de presencia de taza en la máquina de café. El front-end preguntará si hay una taza colocada correctamente antes de iniciar la ejecución. Simular tanto la respuesta afirmativa como la negativa (idealmente 1 de cada 5 o 10 veces debería darse la respuesta negativa sin necesidad de modificar el código del back-end)

**Req(21).** Para el Req(14), simular el apagado de luces en modo “ahorro de energía”

**Req(22).** Para el Req(15), si la alarma está conectada y el usuario realiza alguna de las actividades listadas se deberá enviar un mensaje al front-end indicando que la alarma está “en alerta” (sonando).

**Req(23).** Para el requisito Req(11) simular (de manera aleatoria) algún evento que active la alarma. A discreción del desarrollador.

**Req(24).** Simular el comportamiento de los habitantes de la vivienda mediante el funcionamiento de actuadores de manera aleatoria. Cualquier cambio deberá ser notificado al front-end para que actualice el estado del actuador afectado.

### 9.4.3 Auction site

El proyecto "El juego de la subasta" consiste en la creación de un sistema informático que permita realizar subastas de cualquier producto deseado a través de un portal de Internet. A través del mismo, los usuarios internos podrán dar de alta productos en el sistema e indicar en qué momento se inicia la subasta del mismo. Por otra parte los usuarios externos del sistema, podrán buscar estos productos y pujar por el que deseen, y si no ganan dicha subasta podrán acumular puntos por cada puja realizada los cuales podrán utilizar para comprar productos que serán ofertados a través del sistema. A parte de estas funcionalidades, el usuario externo podrá editar su perfil personal que los demás usuarios podrán visualizar.

#### 9.4.3.1 Alta Usuario Externo

**Requisito 1:** la interfaz general del sistema mostrará en todo momento un enlace a la pantalla de alta de usuario a todo aquel usuario anónimo. Este enlace se ubicará en la esquina superior derecha de la pantalla facilitando su fácil ubicación. Este enlace no se mostrará en caso de que el usuario esté ya identificado en el sistema. Al posicionar el ratón sobre el enlace indicado previamente, aparecerá un "tooltip" que le mostrara al usuario un pequeño bloque con las ventajas que conlleva estar dado de alta en el sistema (como son la posibilidad de participar en pujas, adquirir productos por puntos acumulados y tener un perfil propio).

**Requisito 2:** el formulario a llenar para darse de alta en el sistema contendrá los campos nombre, apellidos, dirección, código postal, ciudad, país, fecha de nacimiento, teléfono, correo electrónico, contraseña y nombre de usuario. Se le pedirá al usuario comprobar el correo electrónico y la contraseña con dos campos más en el formulario. Se le mostrará al usuario las políticas y reglas del JDSL y este deberá marcar que está de acuerdo con las mismas antes de poder continuar con el alta. Una vez completado el formulario de alta de usuario y presionado el botón de continuar, se le informará, con un mensaje en el centro de la pantalla de un color que resalte sobre los demás, que se le ha enviado un correo electrónico el cual debe abrir y pinchar en el enlace de confirmación de su alta. Una vez pinchado el enlace se le llevará a una pantalla donde se le pedirá introducir su correo electrónico y contraseña nueva vez para terminar con el proceso de alta en el sistema.

**Requisito 3:** el sistema verificará que el correo electrónico sea una dirección válida (de la forma abc@dominio.com). Se comprobará que el correo electrónico ya no esté siendo utilizado por otro usuario. De ser así se le mostrará un mensaje al usuario, de un color que resalte sobre el resto de la pantalla próximo a la entrada de texto, de que este está siendo utilizado y no se puede continuar utilizando este (lo mismo ocurrirá con el nombre de usuario). También próximo a este mensaje, se le mostrará una lista de sugerencias de nombres de usuario que no están siendo utilizados en el sistema. Esta lista de sugerencias se construirá a partir del nombre de usuario escrito previamente agregándole números o cambiándole caracteres para así hacerlo único.

**Requisito 4:** el sistema verificará que la contraseña cumpla con un mínimo de nivel de seguridad. La misma deberá poseer por lo menos un número y un carácter especial y no tener menos de 6 caracteres. En caso de que la contraseña escrita no cumpla con el mínimo de seguridad se le mostrará un mensaje de error al usuario.

**Requisito 5:** se generará de forma automática un identificador único para cada uno de los usuarios externos dados de alta en el sistema.

**Requisito 6:** luego de finalizada el alta del usuario se le indicará al mismo que puede editar su perfil personal y se le mostrará un enlace a la pantalla de edición de perfil. Se le mostrarán también enlaces a las secciones del sistema donde puede acceder así como también mensajes informativos y de ayuda sobre qué puede hacer a continuación.

**Requisito 7:** al lado del botón de continuar/finalizar alta de usuario existirá otro botón que permitirá cancelar el proceso de alta. Al hacer clic sobre el mismo el usuario será redirigido a la pantalla donde se encontraba antes.

#### 9.4.3.2 Identificación

**Requisito 8:** la interfaz general del sistema mostrará en todo momento un enlace a la pantalla de identificación en el sistema (“login”). Este enlace se ubicará en la esquina superior derecha de la pantalla facilitando su ubicación. Al dejar el ratón sobre el enlace aparecerá un “tooltip” que le mostrará al usuario un pequeño bloque donde se detallará que se puede realizar al estar identificado en el sistema. Este enlace se sustituirá, en caso de que el usuario ya esté identificado en el sistema, por uno que cierre la sesión actual en el sistema (“logout”). Al dejar el ratón sobre el enlace aparecerá un “tooltip” que indicará que al hacer clic sobre el mismo se cerrará la sesión actual y ya no podrá acceder a todas las funcionalidades del sistema.

**Requisito 9:** en la pantalla de identificación, el sistema le solicitará el correo electrónico y contraseña de acceso. En caso de que los datos introducidos sean incorrectos se le indicará al usuario que debe intentarlo de nuevo y se le ofrecerá la opción de recuperación de contraseña. En caso de que los datos introducidos sean correctos se redirigirá el usuario a la pantalla donde se encontraba antes de iniciar el proceso de identificación y se le habilitarán las opciones correspondientes a su rol. En caso de intentar introducir un usuario y contraseña incorrectos tres veces, la pantalla mostrará una imagen con una palabra que el usuario deberá introducir en una nueva entrada de texto para confirmar que es un usuario humano y no un “bot” el que intenta acceder el sistema y así evitar posibles ataques que busquen averiguar la contraseña de un usuario.

**Requisito 10:** el sistema permitirá la recuperación de contraseña de usuario (req. 5) con un enlace que se ubicará debajo de la entrada de texto de la misma. En esta pantalla el usuario deberá especificar el correo electrónico para el cual quiere recuperar la contraseña. El sistema enviará un correo electrónico a la dirección especificada donde el usuario tendrá que pinchar en un enlace que lo llevará a una nueva pantalla donde deberá especificar la nueva contraseña de acceso y confirmarla. Se le informará al usuario con una letra y fuente que resalten sobre las demás de la pantalla de que ha sido enviado este correo y de los pasos a seguir para finalizar con la recuperación de la contraseña.

**Requisito 11:** en la pantalla de identificación se mostrará un enlace a la pantalla de alta de usuario. Este enlace desplegará el mismo “tooltip” que el descrito en el Requisito 1.

**Requisito 12:** al hacer clic en el botón de acceder se mostrara un “progress bar” en el centro de la pantalla (se oscurecerá la ventana y el “progress” resaltará sobre los demás elementos de la ventana) para indicarle al usuario de que esta operación puede tardar varios segundos

#### 9.4.3.3 Edición de Perfil

**Requisito 13:** el sistema permitirá al usuario externo disponer de un perfil personal el cual podrá ser visualizado por los demás usuarios del sistema. La pantalla de edición de perfil permitirá modificar los siguientes campos: descripción personal, intereses, productos que le gustaría adquirir y foto de perfil. Existirá un enlace próximo a la descripción personal que al hacer clic sobre el mismo se levantará un “popup” donde se le explicará al usuario que podría contener el perfil y que no debe y en caso de que el usuario violara estas normas su perfil no sería aprobado (ej. malas palabras, pornografía, etc). La selección de intereses será una lista previamente definida de posibles intereses en productos del sistema (ej. televisores, equipos de sonido, ordenadores, etc.) de la cual el usuario podrá elegir tantas opciones como desee. Esta lista puede ser utilizada en el futuro para enviarle anuncios al usuario y realizarle ofertas sobre los productos relacionados. Al hacer clic en el botón de finalizar la edición del perfil, se

le mostrará una ventana emergente (“popup”) de confirmación al usuario donde podrá seguir con la modificación o cancelar la misma si así lo desea.

**Requisito 14:** existirá una página principal del usuario que contendrá aquellos enlaces que tienen que ver directamente con el usuario como son: cambio de contraseña, historial de subastas y compras, edición de perfil, contacto con la administración de JDLS y ayuda. También, la interfaz general del sistema mostrará en todo momento un enlace a esta pantalla en la esquina superior derecha de la misma para facilitar su ubicación. Al dejar el ratón sobre cualquier de los enlaces aparecerá un “tooltip” que le mostrará al usuario un pequeño bloque con una breve descripción de la pantalla a la que redirige dicho enlace.

**Requisito 15:** en la pantalla especificada en el Requisito 14 también existirá un enlace a la pantalla de configuración del sistema (opciones del sistema específicas para el usuario de lugar). En esta pantalla se podrán activar o desactivar las notificaciones del sistema por correo (mensajes del administrador, propaganda, etc.) como también las notificaciones de subastas ganadas o perdidas. En esta pantalla también existirá la opción de habilitar o deshabilitar la recepción de mensajes privados. Existirá un espacio reservado en la interfaz general del sistema (al lado del nombre de usuario) donde se indicará que la recepción de mensajes privados se encuentra desactivada.

#### 9.4.3.4 Visualización de Perfil

**Requisito 16:** el sistema permitirá a todos los usuarios visualizar el perfil de un usuario externo en cualquier momento. Siempre que aparezca el nombre de usuario de un individuo en la pantalla (ej. usuario ganador de una subasta o usuario comprador de un producto en subasta) este será un enlace que llevará directo al perfil del usuario en cuestión.

**Requisito 17:** la pantalla del perfil de usuario mostrará los campos descripción personal, intereses, productos que le gustaría adquirir y foto de perfil. También permitirá ver el historial de subastas en la que ha participado el usuario, que productos ha ganado y cuales ha adquirido por puntos acumulados. Estos últimos dos puntos se visualizarán en forma de lista y al pinchar en una de las entradas el usuario podrá ir directamente a la pantalla de esta subasta o producto en oferta. Este perfil deberá estar actualizado en tiempo real, es decir que si un usuario está participando en una subasta, su perfil se debe actualizar al momento en que este realice alguna puja. En este momento, al usuario que esté visualizando el perfil del usuario que ha realizado la puja, le aparecerá un mensaje en un color y fuente que resalte sobre los demás elementos de la pantalla indicándole que este ha realizado una puja y en que subasta.

**Requisito 18:** en la pantalla del perfil del usuario existirá un enlace que permitirá a otro usuario externo enviar un mensaje privado al usuario dueño del perfil en cuestión. Este enlace debe estar deshabilitado si el usuario en cuestión no tiene la opción de recibir mensajes privados activada. Al pinchar en el enlace se levantará una ventana emergente (popup) que solicitará ingresar un asunto y cuerpo de mensaje el cual llegará al correo del usuario de lugar. Este enlace solo aparecerá siempre y cuando el usuario tenga habilitada la opción de permitir mensajes privados en su configuración personal y que el usuario que visualiza el perfil se encuentre identificado en el sistema.

#### 9.4.3.5 Búsqueda de Productos

**Requisito 19:** la interfaz general del sistema dispondrá de una entrada de texto donde cualquier usuario del sistema podrá realizar una búsqueda general. Esta entrada estará ubicada en la parte superior izquierda de la pantalla para su fácil ubicación. Al escribir un texto en esta entrada y presionar la tecla “enter” o hacer clic en el botón próximo a la misma, el usuario será redirigido a la pantalla de resultados de la búsqueda. Al ir introduciendo texto en la entrada aparecerá una lista con productos sugerencia al término introducido (ej. si se

escribe Sams, la lista mostrará los productos que contengan Sams en su nombre o sean de una marca que contenga Sams, como lo es Samsung).

**Requisito 20:** en la pantalla de búsqueda de productos se podrán realizar búsquedas avanzadas de productos, tanto en subasta como en oferta. El primer paso para realizar una búsqueda es escribir en una entrada de texto el nombre o parte del nombre de un producto a buscar si así se desea. Al ir escribiendo en la entrada de texto aparecerá una sugerencia como la descrita en el Requisito 17. El resultado de la búsqueda se podrá filtrar por categoría del producto, por tipo (subasta u oferta), por tiempo restante de la subasta, por cantidad de puntos máxima para obtener el producto en oferta, por cantidad de pujas en una subasta, por fecha de creación de la subasta/oferta, y por todas aquellas características específicas de un producto (ej. de la categoría televisores se podría filtrar el tamaño de pantalla, el contraste, el peso, la tecnología, etc.).

**Requisito 21:** los resultados de la búsqueda se mostrarán en forma de tabla debajo del panel de búsqueda y cada fila de la misma contendrá las columnas de nombre, descripción, imagen principal, tiempo restante de la subasta o puntos necesario para adquirir el producto en oferta y fecha de creación. Estos resultados podrán ordenarse haciendo clic en las cabeceras de cada columna. Esta lista permitirá al usuario acceder a la subasta u oferta en cuestión haciendo clic sobre el nombre del producto.

**Requisito 22:** el usuario deberá poder saber cuál es el filtro aplicado a la búsqueda actual y cuál es el criterio de ordenación de los resultados en todo momento. Para ello se mostrará un pequeño resumen con los filtros aplicados y al lado de la columna por la que se esté ordenando en la tabla de resultados aparecerá una imagen indicando el orden de los registros (flecha hacia arriba indicando orden ascendente o hacia abajo indicando orden descendente).

**Requisito 23:** en la visualización de un producto en subasta o en oferta existirá un botón visible para todos los usuarios externos donde podrán agregar dicho producto a su lista de “favoritos” (este botón estará ubicado próximo al nombre del producto). Esta lista podrá ser visualizada en una pantalla cuyo enlace se encontrará en la pantalla especificada en el Requisito 14 y desde la misma se podrá acceder directamente al producto correspondiente al favorito seleccionado. Si una subasta o un producto en oferta se han eliminado del sistema desaparecerán automáticamente de la lista de favoritos de los usuarios que lo tuviesen agregado.

#### 9.4.3.6 Puja en Subasta

**Requisito 24:** en la pantalla de un producto en subasta un usuario externo podrá pujar en cualquier momento por dicho producto siempre y cuando el tiempo de la subasta no haya finalizado y dicho usuario tenga puntos suficientes para realizar la puja. La puja se llevará a cabo pinchando en un botón que estará ubicado en la parte superior derecha de la página del producto (este botón deberá ser de tamaño considerable y de un color que lo haga resaltar sobre los demás elementos de la pantalla) debajo del cronómetro de la subasta que marcará el tiempo restante de la misma. Al pujar el usuario se verá reflejado como actual dueño del producto y su nombre de usuario le aparecerá a todos los demás usuarios que estén visualizando dicha subasta en tiempo real. El nombre de usuario del usuario que esté ganando se mostrará debajo del botón de pujar en un color y fuente que resalte sobre los demás elementos de la pantalla.

**Requisito 25:** si al momento de hacer clic en el botón de pujar por un producto en subasta el usuario no cuenta con puntos suficientes para realizar la puja, se le mostrará un mensaje en una ventana emergente indicándole de que debe recargar sus puntos antes de poder continuar y se le mostrará un enlace para ir directamente a la pantalla de recarga de puntos.

**Requisito 26:** cada vez que un usuario puje en una subasta el tiempo de restante de la subasta aumentará una cantidad de segundos a determinar. Una vez finalizada la subasta porque nadie más ha pujado en la misma, a todo usuario que esté visualizando la página del producto recibirá un mensaje mediante una ventana emergente de que la misma ha terminado e indicándole quien ha sido el ganador. También se indicará cuantas pujas se realizaron por el producto. El botón para pujar se bloqueará y donde antes salía el tiempo restante de la subasta saldrá un mensaje indicando quien ha sido el ganador y con cuantas pujas.

**Requisito 27:** al realizar una puja al usuario se le restará un punto de los puntos para puja que ha comprado previamente y se le sumará uno a la cantidad de puntos acumulados para canjear por productos de los que dispone.

Requisito 28: existirá una sección de la pantalla en donde se mostrarán los puntos para puja de los cuales dispone el usuario en todo momento y en caso de este realizar una puja esta sección se actualizará automáticamente. También se mostrarán los puntos acumulados para adquirir productos en oferta.

#### 9.4.3.7 2.3.2.7 Compra de Producto en Oferta

**Requisito 29:** en la pantalla de un producto en oferta un usuario externo podrá adquirir el producto si dispone de puntos acumulados suficientes y si el tiempo de la oferta no ha terminado. Existirá un botón ubicado al lado derecho del nombre del producto y al pincharle se le mostrará una ventana emergente al usuario donde confirmará la compra del producto. Si el usuario dispone de puntos suficientes para realizar la compra se le redirigirá a la pantalla de finalización de la misma de caso contrario en esta misma ventana se le indicará que no es posible realizar la compra de este producto. Al confirmar la compra en esta pantalla al usuario se le restarán los puntos necesarios automáticamente de su acumulado. El usuario también tendrá la opción de cancelar la compra cuando lo desee.

**Requisito 30:** si al momento de hacer clic en el botón de comprar un producto en oferta el usuario no cuenta con puntos acumulados suficientes para realizar la compra, se le mostrará un mensaje en una ventana emergente indicándole de que debe seguir participando en subastas para acumular más puntos y se le mostrará un enlace para ir directamente a la pantalla de bienvenida del sistema.

**Requisito 31:** cuando un usuario externo adquiere un producto, a los demás usuarios que estuvieran viendo la página del producto en cuestión se le notificará mediante un mensaje de un color que resalte sobre el resto de la pantalla de que el producto ya ha sido adquirido por otro usuario y se le deshabilitará el botón de compra.