ELSEVIER

Available online at www.sciencedirect.com

ScienceDirect

Procedia Technology 9 (2013) 236 – 246

Procedia Technology

CENTERIS 2013 - Conference on ENTERprise Information Systems / PRojMAN 2013 - International Conference on Project MANagement / HCIST 2013 - International Conference on Health and Social Care Information Systems and Technologies

Reuse of a usability functionality implementation in web applications

Francy D. Rodríguez[a]*, Silvia T. Acuña[b]

[a]*Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo s/n Madrid, Boadilla del Monte 28660, España*
[b]*Departamento de Ingeniería Informática, Universidad Autónoma de Madrid, Calle Francisco Tomás y Valiente 11, Madrid 28049, España*

Abstract

Software system usability is recognized as a quality attribute related not only to the user interface but also to applications design. In this paper we analyse the feasibility of designing and programming reusable solutions for implementing usability features that have a major impact on design. We develop case studies to find common application scenarios, responsibilities, classes, methods, attributes and chunks of code, which we use to propose reusable solutions specified as patterns. In this paper we report the results for the progress feedback usability functionality.

Keywords: Software engineering; Reuse; Usability; Design pattern; Programming pattern

* Corresponding author. *E-mail address:* fd.rodriguez@alumnos.upm.es.

1. Introduction

Usability is a software application quality attribute, and a potentially critical feature of highly interactive systems [1][9]. Usable systems have been proven to generate sizeable savings and increased benefits [10][13][8]. On these grounds, more and more software system developers are taking usability features into account. The human-computer interaction (HCI) field has addressed usability at length and proposed recommendations and techniques for building usable interactive systems.

The adoption of usability in software engineering (SE) has been a staged process. Initially, usability was viewed as a non-functional requirement related exclusively to the user interface (UI), meaning that it was tackled towards the end of the development process and was implemented using approaches that separated UI functionality from the core system functionality [2]. One example of this approach is the model view controller (MVC) pattern, which is widely used to develop software and especially web applications. Subsequent studies argued that usability aspects generate static and dynamic constraints on software components [14] that the strategy of separation is not a satisfactory way of producing a usable system and that there are usability aspects that should be addressed as of the early development phases, particularly at system architecture design time [3]. They looked at how usability interacted with system functionality and not just with the UI. Juristo et al. [11] provided empirical evidence that some usability features are connected to software design, identified usability features that have a major impact on design and measured their impact on real applications. The requirements elicitation field has also examined usability and developed guidelines for eliciting usability functionalities [12]. These guidelines set out a series of questions, based on the HCI recommendations for each usability feature, for discussion with users.

So, plenty of research has been conducted in SE on the inclusion of usability features into software products at different stages of the development process. They have come up solutions that can be easily reused and popularized throughout the developer community, for which purpose they are often specified as patterns. The above studies focus on the early phases of the development process: requirements elicitation [12] and architecture design [2][3][4][16][11]. In this paper we focus on two of the later development cycle phases: detailed design and implementation. We propose detailed design and programming patterns for three different languages: VB.NET, Java and PHP 5. Programming patterns are detailed patterns that describe how to implement particular tasks using a specific programming language and make use of the language features to develop parts of components or relationships between components [6].

In this paper we propose a reusable solution for a usability functionality with a major impact on design: progress feedback (PF). The PF functionality focuses on providing users with verbal or graphical signals of the progress of system tasks. PF should advise users of when a task finishes and check whether the user can or cannot execute other tasks simultaneously. The proposed solutions are based on the development of case studies of real web applications.

This paper is structured as follows. Section 2 describes the research method used in this paper and introduces the developed case studies. Section 3 presents a two-part solution. First it introduces the application scenarios discovered for the PF usability functionality and then it outlines the solution specified as programming patterns. Section 4 describes the evaluation of the solution on another two case studies. Section 5 analyses related work proposing the use of patterns for incorporating usability functionalities into software development. Finally, Section 6 outlines the conclusions and future work.

2. Case study-based search of a reusable solution

In this study we have used a three-phase inductive research method, implementing case studies to infer a general-purpose solution. The three developed case studies are interactive web applications based on requirements for real systems. The first is an indicator administration system for creating indicators and simple data, classifying, querying and importing data. This system was developed in PHP 5[†] and has a MySql database. The second case study is a web system for generating payment variables, which updates and manages payroll information. The system was built in VB.NET and has a Microsoft SQL database. The third case is a healthy diet subscriber web system, managing data on subscriber state of health, and including purchase and delivery options. The system was built in Java and has a PostgreSql database.

The web systems were built during phase 1 of the research, assuring that the functionality associated with the PF usability feature was implemented on top of the core system functionality. The elements related to the usability functionality were marked in each artefact generated during the development process. During phase 2, we identified the elements related to the usability functionality implementation that each artefact had in common and established which elements could be reused. In research phase 3, we specified the results as patterns. We propose a single design which is implemented differently for each of the three programming languages: PHP 5, VB.NET and Java. The proposed design and the implemented codes are combined to specify programming patterns. As part of phase 3, we also extracted common code snippets, as a first step towards building a components library for the usability functionality. Finally, two independent developers applied the proposed solution to another two case studies.

3. Description of the solution

The classical waterfall development model ─requirements elicitation, analysis, design and implementation─ was used to develop the case studies. We chose the waterfall model to develop the case studies because it is based on a set of stable requirements, is easy to use and each phase ends with outputs that can be compared later. In phase 1, existing guidelines, shown in the web annex[‡], were used to elicit the usability functionality. Based on responses to the guideline questions, we integrated the usability functionality into the systems requirements, usability functionality and the core system functionality were specified separately in the final requirements document. Usability functionality was also singled out in all the artefacts generated during the other development phases. At the end of the development process, we analysed the elements used to implement the usability functionality in search of commonalities that could be used to design a reusable solution.

We concluded from the analysis of the results of the requirements elicitation phase that there is more than one option. These options depend on user responses to the elicitation questions and may or may not apply depending on the application under development. Each option is the result of a combination of user responses and has different design implications. We refer to the different options as scenarios. Section 3.1 details how we arrived at the different application scenarios. Section 3.2 describes the process enacted to identify and specify reusable design and implementation elements as programming patterns.

3.1. Application scenarios

The context for PF functionality implementation is when a system process is likely to block the UI for longer than two seconds. The elicitation guide raises the following issues for discussion: Which tasks are likely to take longer than two seconds? Which of the identified tasks are critical? How will the user be informed that the process has finished? How will the user be informed about the progress of each task? What information do they need in each case?

† Early PHP versions were not object oriented; however, the functionalities required for programming using classes and objects were implemented as of version 5.

‡ http://www.grise.upm.es/sites/extras/7/Usability_Elicitation_Pattern_PF.pdf

The elicitation guide also summarizes the HCI recommendations on which the above issues are based. These recommendations provide more details about aspects to be taken into account in the implementation. For example, one HCI recommendation states that the information to be provided should include the proportion of the operation that has been completed and/or the time remaining to finish the task. This recommendation calls for major web application design and implementation decisions: the use of asynchronous processes to discover the progress of a task executed on a server or the division of a process into more than one task to be executed on the client side and in order to monitor progress on a task-by-task basis.

Each combination of responses to the questions in the elicitation guide generates an application scenario that may or may not be displayed depending on the software system in question. Some scenarios call for changes to the design and implementation decisions. In the following we illustrate the relationship between the elicitation guide issues, associated HCI recommendations, possible application scenarios and their technological implications:

Question: Which tasks are likely to take longer than two seconds?
Software engineers should answer this question after determining the system requirements and which development tool is to be used.

Question: Which of the identified tasks are critical?
Associated HCI recommendation: If the process is critical, users should not be allowed to do anything else until the task is completed. If the task is not critical and takes over 5 seconds, users should be allowed to run another task if they so wish.
Scenarios:
1. The process is critical. Users are not allowed to perform another operation.
2. The process is not critical and takes less than 5 seconds. Users are not allowed to perform another operation.
3. The process is not critical and takes more than 5 seconds. Users are allowed to perform another operation if they so wish.

Technological implications: Scenario 3, where the users are allowed to perform two operations simultaneously, would, in web applications, imply using asynchronous processes and controlling all possible events on the navigable pages in order to monitor on-going processes. It would also call for a server-side mechanism for informing when a process has finished.

Question: How will the user be informed that a process has finished?
Scenarios:
1. Display and automatically close a message reporting the results. The progress indicator closes.
2. Display a message until it is closed by the user.
3. Display and automatically close a message on the progress indicator.
4. Display a message on the progress indicator until it is closed by the user.
5. Display a result in place of a message, for example, in response to queries.

Technological implications: A mechanism should be implemented to monitor the process state if the usability functionality is responsible for informing when the process has finished.

Question: How will the user be informed about the progress of each operation?
Associated HCI recommendation: About remaining time: If the time can be calculated, use a time-remaining progress indicator or a proportion-completed progress indicator; if the time cannot be calculated but the process has identifiable phases or tasks, use a progress checklist; if neither of these possibilities exist, then indicate the number of units processed, and if no quantities are known, use an indeterminate progress indicator.
Scenarios:
1. Display a time-remaining progress indicator.
2. Display a proportion-completed progress indicator.
3. Display a progress checklist.
4. Display a message reporting the number of processed units.
5. Display an indeterminate progress indicator.
6. Display a time-remaining progress indicator and number of processed units

7. Display a proportion-completed progress indicator and number of processed units

8. Display a progress checklist and the progress of each task on the checklist using an indicator of time, proportion or units.

Technological implications: Single server-side processes that require the provision of progress information will require the use of asynchronous processes where different threads handle the core process and a separate progress monitor and update process. A mechanism for checking and reporting progress should be added to the latter process. Another option would be to subdivide a task into n processes, where process execution would be controlled from the client side and progress reported as the number of processes completed. A combination of the two options is another possibility.

Question: Which information is considered necessary in each case?

Associated HCI recommendation: The progress of a task will be displayed verbally or graphically, and, whenever possible, the following information should be provided: on-going operation, proportion of the operation that has been completed, time remaining until operation finishes.

Scenarios:

1. Display process name.
2. Display lower and upper bound of values, e.g., 0% to 100%, task 1 to n, 0 to x records, etc.
3. Display the current progress value.
4. Display a description of the phase or task as part of the process as a whole.

Technological implications: The graphical component of the progress indicator should be able to be configured to display, depending on the available information, each and every one of four options specified by the scenarios and their possible combinations.

Associated HCI recommendation: The indicator should inform users of how to cancel an operation if it is estimated to take longer than 10 seconds.

Scenarios:

1. The operation cannot be cancelled. The actions open to the user depend exclusively on whether or not the task is critical.
2. The operation can be cancelled. Display a Cancel option.

Technological implications: If a task is cancellable, the functionality required to terminate the process without rendering the system insecure or unstable should be implemented.

The technological implications inferred above result in a number of conditions that significantly modify the system design and implementation, including task criticality, the type of available progress information, process cancellability, or usability functionality responsibility for informing whether or not a process has finished.

Apart from the above implications, there is another variable to be taken into account: technology. The design will be different depending on whether or not the technology is able to manage multiple threads of execution, i.e., multithreaded or single-threaded technology. A language that supports multithreading can run several processes simultaneously, each with its own control flow. Using multithreaded technology, separate server-side processes can be used to update the progress of a process whenever necessary. Using single threaded technology, it will not be possible to execute another task to monitor progress until the core process has finished. One option in this case is to use an indeterminate progress indicator. The other alternative would be to divide the process into several tasks to be executed one at a time and display a task-by-task progress checklist.

In sum, we identified 12 application scenarios for the PF usability functionality. The scenarios are conditioned by possible responses to the elicitation questions and by the type of technology that is used. Fig. 1 shows a tree with all the identified scenarios. Note that the nodes nearest to the tree root are related to the responses to the elicitation questions, whereas the leaf nodes are dependent on the features of the technology.

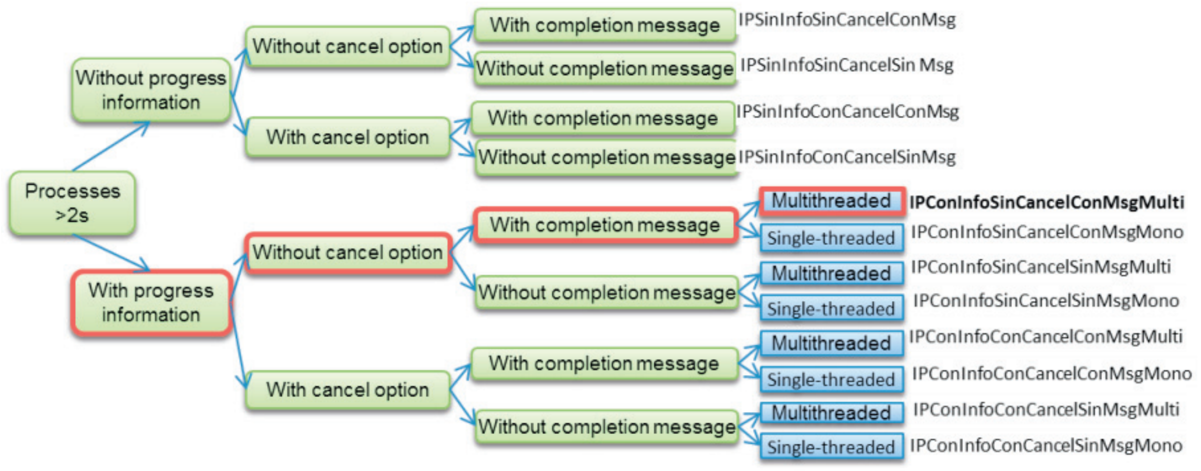
Processes >2s
Without progress information
With progress information
Without cancel option
With cancel option
Without cancel option
With cancel option
With completion message
Without completion message
With completion message
Without completion message
With completion message
Without completion message
With completion message
Without completion message
Multithreaded
Single-threaded
Multithreaded
Single-threaded
Multithreaded
Single-threaded
Multithreaded
Single-threaded
IPSinInfoSinCancelConMsg
IPSinInfoSinCancelSin Msg
IPSinInfoConCancelConMsg
IPSinInfoConCancelSinMsg
IPConInfoSinCancelConMsgMulti
IPConInfoSinCancelConMsgMono
IPConInfoSinCancelSinMsgMulti
IPConInfoSinCancelSinMsgMono
IPConInfoConCancelConMsgMulti
IPConInfoConCancelConMsgMono
IPConInfoConCancelSinMsgMulti
IPConInfoConCancelSinMsgMono

Fig. 1. Application scenarios tree for the progress feedback functionality

Each scenario has a name, and its operation has been documented using sequence diagrams. For example, IPConInfoSinCancelConMsgMultihilo denotes the following scenario: process with progress information, which cannot be cancelled, requires a completion message and is implemented using multithreaded technology. The sequence diagrams of all the scenarios are illustrated in the web annex[§].

3.2. Specification of the solution as programming patterns

We used the application scenarios singled out for all three case studies to identify the common responsibilities necessary to cover the functionality associated with the PF usability feature. The identified responsibilities are:

•Check whether the process is still active, provided progress information is available and multithreaded technology is used.
•Generate a server-side mechanism to update the active process and report its progress.
•Create a cyclical process to monitor the progress of an operation until it finishes.
•Display the right progress indicator depending on the available information.
•Inform the user when the operation finishes.
•Display completion message and close progress indicator.

We found that similar classes were used in all three case studies to cover the above responsibilities, which we merged into five classes. Fig 2 shows the final class diagram. The responsibilities covered by each class are as follows.

ProgressFeedbackUI. This class displays the right progress indicator depending on the available information ─time, percentage, processed units, completed tasks─, or an indeterminate progress indicator when there is no information. It draws the progress indicator on the UI according to the supplied parameters: title, size, process name, task name, mode, initial value, etc. It updates the values displayed throughout. The position of the progress indicator on the UI can be changed. It informs the user when a process finishes. It displays the close or cancel button and a completion message as appropriate. It closes the progress indicator.

[§] http://www.grise.upm.es/sites/extras/7/Escenarios_MU_PF.pdf

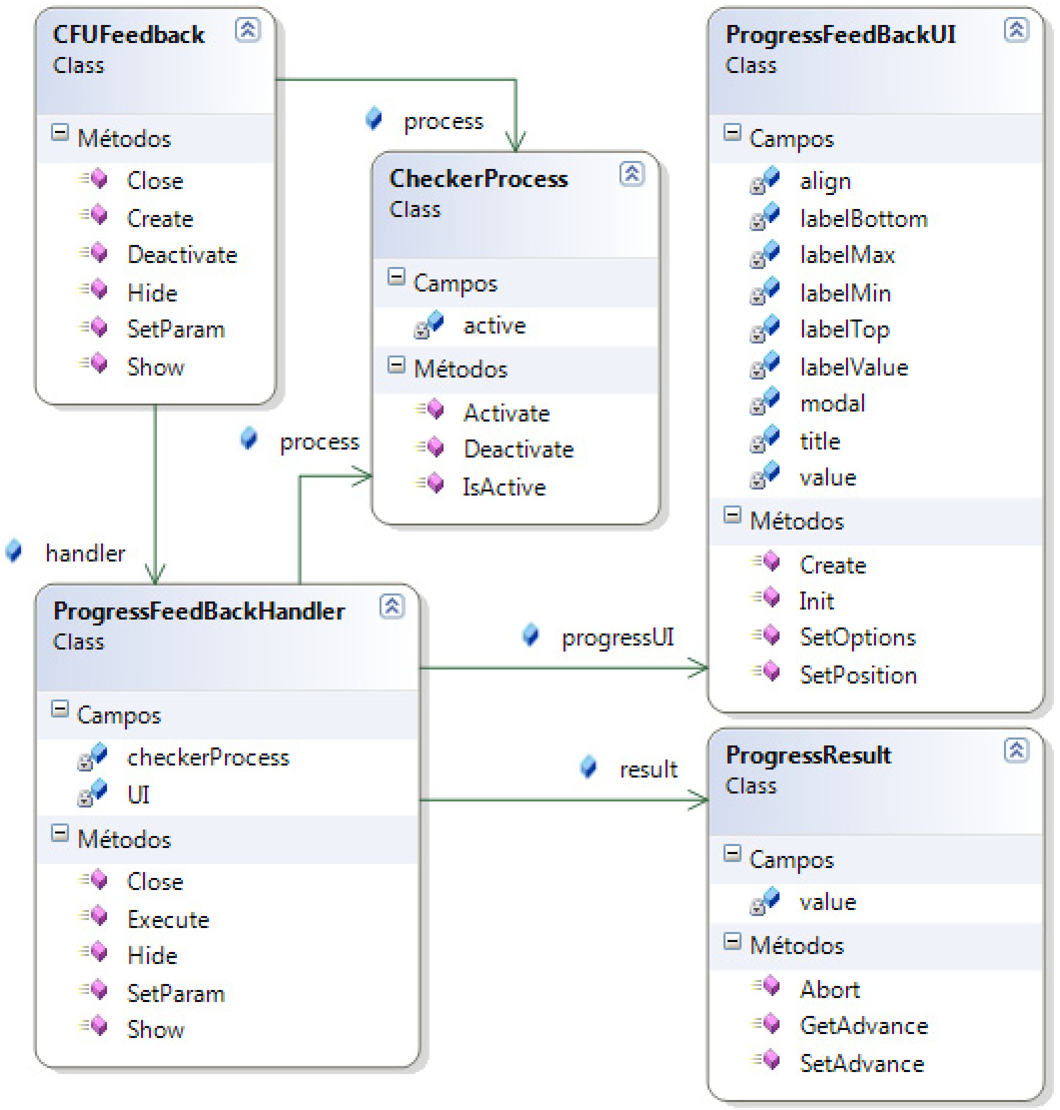

Fig. 2. Progress feedback usability functionality class diagram

CheckerProcess. This class checks whether a process is still active. It establishes whether to continue monitoring progress and displaying the progress indicator.

ProgressFeedbackHandler. This class handles the user-generated events and server responses. It launches the right options depending on the event and the supplied information. It also weighs up the possibility of displaying more than one progress indicator at the same time. It is responsible for creating and updating ProgressFeedbackHandler instances in order to display and update information on screen. It handles cyclical processes that check the progress value of an object in the server once every given time interval.

ProgressResult. This class is a server-side component that updates the process progress information throughout a session. Its function is to update and deliver the process progress information as and when requested.

CFUFeedback. This class is used to separate system functionality and PF functionality. It is responsible for distributing requests to usability functionality components, separately from the application functionality.

Because we are dealing with web applications, we have to make a distinction between client-side and server-side code. Client-side code is built in the same script language for all three use cases: JavaScript. The same code is thus applicable to all three cases. The server-side code, on the other hand, is technology dependent and therefore different for each of the proposed designs.

At client level, we managed to design one piece of JavaScript code that was reusable across all the web applications irrespective of the implemented technology. Table 1 shows the JavaScript code of the CheckerProcess component, which is the same for all three case studies. For the proposed programming patterns, see the web annexes: VB.NET[**], Java[††] and PHP 5[‡‡].

Table 1. CheckerProcess component code

```
/* ------------------------------------------------//Class: CheckerProcess
//Description: Updates information on whether the process is active ---------------------*/
//The process is active.
function CheckerProcess() { this.active = true;}
CheckerProcess.prototype.Activate = function(){ this.active = true;}
CheckerProcess.prototype.Deactivate =function(){ this.active = false; }
CheckerProcess.prototype.IsActive = function() { return this.active; }
```

4. Evaluation

The proposed solution was used by two independent developers to implement two additional web applications. Each developer developed a different case study as part of their Master theses. They used two of the languages for which programming patterns were provided in this research: Java and VB.NET. Apart from the core functionality of application, the developers were instructed to include usability functionality. They used the same elicitation guidelines as we used in this paper to elicit requirements, as well as the research results reported here: application scenarios, design pattern and implementation pattern.

In one of the cases, it was possible to use the programming pattern, that is, the developer used the proposed design and supplied code. In other case, the developer was able to use only the detailed design and the implementation guide, because the code was incompatible with the technology used. Note, however, that the implementation guide supplied with both the design pattern and the implementation pattern is useful for implementing the usability functionality even if another technology is used.

The developers found the application scenarios to be useful for understanding the implications of the implementation of the usability functionality. In the requirements elicitation and specification phase, the scenarios helped developers to check that they had taken into account all the possible cases in which the usability functionality was, according to system requirements, applicable. In the initial design phase, it helped them understand how the inclusion of the usability functionality would affect the implementation of the application functionalities. Finally, in the coding phase, it substantially modified how the processes requiring

** http://www.grise.upm.es/sites/extras/7/PP_PF_VB_NET.pdf
†† http://www.grise.upm.es/sites/extras/7/PP_PF_Java.pdf
‡‡ http://www.grise.upm.es/sites/extras/7/PP_PF_PHP.pdf

progress feedback were programmed. It took developers a long time to understand the code proposed in the pattern, partly because they were inexperienced in asynchronous programming. The supplied code was usable for VB.Net, but not for Java because the technology and framework (Jquery and Java Server Faces) used in the pattern was incompatible with the framework (Struts) used by the developer.

5. Related work

After it had been established that some usability features have an impact on the core functionality of software systems, numerous studies were conducted to provide software engineers with methodologies, techniques and tools to effectively and efficiently incorporate usability throughout the development cycle. Patterns are a widely accepted method for formulating reusable solutions within the software development community and are useful for transmitting good practice in a familiar format and language [6]. This has led many authors to use patterns to present the results of their studies and solutions.

Functional usability features should be included as of requirements definition, the very first software development process phase. In [12], Juristo et al. provide guidelines for eliciting usability functionalities. At architecture level, Bass et al. state the need to find strategies other than separating the UI from the application core to generate usable software in [2][4][3]. They identified usability scenarios that should be taken into account from the early development stages and define architectural patterns for usability.

In a later study [5], they applied and even implemented architectural patterns at business level. Based on the results, they proposed a responsibility-based pattern language for architectural patterns supporting usability. Note that although the study covered implementation, Bass et al. opted for a solution at a high level of abstraction. They gave a general description of the responsibilities that the different functional elements should cover, but did not report results at the detailed design or implementation level.

Along the lines of the above research, the STATUS Project [16] studied the relationship between software architecture and usability and presented an approach for improving usability applying a specific design process. The STATUS project proposed the incorporation of usability at the early stages of development and brought forward the evaluation/improvement cycle to high-level design time.

An alternative approach, aspect-oriented programming (AOP), tackled the later software development cycle phases from another angle. In [7], Campos et al. implemented the alerts usability functionality using AOP in order to rule out the interconnection of the core application functionality with usability functionality. In view of how many usability features have been found to have an impact on design, further studies are required in this respect to determine which usability functionalities can be implemented using AOP and which are better modelled as components [15].

In this paper we focus on the later stages of the development cycle and look to provide a solution that will improve developer efficiency and error prevention in the detailed design and implementation stages. To do this, we provide a detailed design and code to implement the design in different languages. We also use application scenarios that detail the functionality covered by the design. We document the code so that it can be reused using either a cut-and-paste strategy or by directly including the library.

6. Conclusions and future work

In this paper we present a reusable solution for implementing the progress feedback usability functionality. We analysed the requirements after examining the usability functionality in three case studies and found that there are multiple application scenarios for this functionality. The application scenarios are generated by combining the responses to specialized elicitation questions for the usability functionality.

We determined that the PF functionality has major implications for the design and implementation of a web application. The software design and implementation has to be substantially modified to comply with the HCI recommendations for this usability feature. One of these implications is the use of multithreaded programming to monitor the progress of an on-going process. One process thread has to be executed to check and display progress, whereas the other executes the core system task. This thread should also be modified to update progress information for monitoring, as well as executing the core task.

Another point to take into account is whether the language to be used supports multithreaded programming. Two of the languages used in our research, VB.NET and Java, support multithreading, but PHP 5 does not. This means that the design has to be adapted in order to provide progress feedback with progress information on a process for PHP 5. For example, a process might be divided into several tasks. Each task would be sent from the client, and progress would be reported in terms of number of tasks finished.

As we are dealing with web applications, the same JavaScript language can be used on the client side. This means that the designed code can be used irrespective of the programming language used for the server. In this case, four of the five proposed classes are located on the client side, the code designed is stored in a single file and works for all three case studies developed in this paper. One of the independent developments used to evaluate the solution also used this code.

Programming patterns are useful in two respects. First they provide a detailed design that is easy to understand and covers a set of functionalities, which, in this case, are described as application scenarios. The pattern sets out a solution indicating how to implement the language-independent design step by step, that is, it describes how to implement the design even if the sample code supplied is not usable. Second, if developers are using the same language, technology and version as specified in the pattern, it can be applied directly adopting a cut-and-paste strategy or by including the entire library.

In the future, we intend to develop new case studies with and without patterns. We aim first to continue evaluating and adding to these preliminary results, and second to find out whether the solution offers real benefits. We also intend to extend the study to other usability functionalities with a high impact on design.

Acknowledgements

This work has been funded by the Spanish Ministry of Science and Innovation Tecnologías para la Replicación y Síntesis de Experimentos en IS (TIN2011-23216) and Go Lite (TIN2011-24139) projects.

References

[1] ISO, 9241-11, Ergonomic Requirements for Office Work with Visual Display Terminals. Part 11: Guidance on Usability, ISO, 1998.
[2] Bass L, John B. Supporting usability through software architecture. *IEEE Computer* 2001;Vol. 34(10) , p. 113-115.
[3] Bass L, John B. Linking usability to software architecture patterns through general scenarios. *Journal of Systems and Software* 2003; Vol. 66(3), p. 187-197.
[4] Bass L, John B, Kates J. Achieving Usability through Software Architecture, Technical Report CMU/SEI-2001-TR-005, Software Eng. Inst., Carnegie Mellon Univ. 2001.
[5] John B, Bass L, Golden E, Stoll P. A Responsibility-Based Pattern Language for Usability-Supporting Architectural Patterns. *EICS*. 2009.
[6] Bushmann F, Meunier R, Rohnert H, Sommerlad P, Stal M. *Pattern - Oriented Software Arquitecture. A system of patterns*. John Wiley & Sons. 1996.
[7] Campos P, Acuña S T, Macías J A. Implementación de Propiedades de Usabilidad con Impacto en el Diseño Mediante la Programación Orientada a Aspectos*, Actas del XI Congreso Internacional de Interacción Persona-Ordenador* 2010; p. 369-378.
[8] Chrush M. Seven Great Myths of Usability. *Interactions* 7, 2000 p. 13-16.
[9] Constantine L, Lockwood L. *Software for use: A practical Guide to the Models and Methods of Usage-centered Design*, Addison Wesley. 1999.
[10] Donahue G. Usability and the Bottom Line, *IEEE Software* 2001; Vol. 18(1), p. 31-37.
[11] Juristo N, Moreno A, Sanchez-Segura M.-I. Analysing the impact of usability on software design, *Journal of System and Software* 2007, Vol. 80, p. 1506-1516.

[12] Juristo N., Moreno A, Sanchez-Segura M-I. Guidelines for Eliciting Usability Functionalities Software Engineering, *IEEE Transactions on Software Engineering* 2007; Vol. 33, p. 744-758.
[13] Nielsen J, Return on Investment for Usability. http://www.nngroup.com/articles/return-on-investment-for-usability/. Last Access 2012.
[14] Perry D, Wolf A.Foundations for the study of software architecture, *ACM Software Engineering Notes* 1992, Vol. 17 (4), p. 40-52
[15] Pinto M, Fuentes L. Aspect-Oriented Modeling of Quality Attributes, *ECSA* 2008; p. 334-337.
[16] STATUS Project. Software Architecture that supports Usability. 2001. http://www.grise.upm.es/rearviewmirror/projects/status/index.html. Last Access: 2012.