Usability through Software Design

Laura Carvajal, Ana M. Moreno, María-Isabel Sánchez-Segura, and Ahmed Seffah

Abstract—Over the past two decades, the HCI community has proposed specific features that software applications should include to overcome some of the most common usability problems. However, incorporating such usability features into software applications may not be a straightforward process for software developers who have not been trained in usability (i.e., determining when, how, and why usability features should been considered). We have defined a set of usability guidelines for software development to help software engineers incorporate particular usability features into their applications. In this paper, we focus on the software design artifacts provided by the guidelines. We detail the structure of the proposed design artifacts and how they should be used according to the software development process and software architecture used in each application. We have tested our guidelines in an academic setting. Preliminary validation shows that the use of the guidelines reduces development time, improves the quality of the resulting designs, and significantly decreases the perceived complexity of the usability features from the developers' perspective.

Index Terms—Software usability, software design, software design patterns

1 Introduction

Usability is defined by ISO as "the capability of a software product to be understood, learned, used, and attractive to the user, when used under specified conditions" [1]. This attribute has gained importance as an integral quality aspect of software development [2]. The benefits of usability for users and software companies have been much highlighted in literature [3], [4], [5]. However, poor usability is the single biggest cause of software application failure in practice [6].

For over a decade, the software engineering (SE) community has been actively pursuing different lines of research targeting the incorporation of usability practices into software development. Part of this research focuses on incorporating human-computer interaction (HCI) techniques and activities into the software development process [2], [7], [1]. On the other hand, software engineers have also long struggled to consistently transform the usability recommendations proposed by the HCI community into software code. This requires dealing with the implications of such recommendations for the software architecture and design. In this respect, special attention has been paid to usability features with implications beyond the user interface (UI), as designing interactive software that explicitly includes usability features with an impact on application logic is not an easy task. Bass et al. [9], [10] pioneered this line and drew the attention of software practitioners by identifying a set of usability scenarios, like displaying status information or providing alternative security mechanisms, which are not straightforward to implement due to their tight relationship with the underlying software architecture. Later, other researchers recognized and worked on this relationship [11], [12], [13]. We discuss other usability recommendations that have an effect that goes far beyond tweaking the user interface and well into software design rework in [13]. For instance, providing the possibility to cancel given commands has implications that need to be addressed by developers during the design of the application logic; storing state information correctly, reverting modifications, notifying the user when the process has finished are some such concerns.

As we discuss later in detail, although many valuable attempts have been made to bridge the gap between usability and software design, the problem is still an open research topic.

Our objective is to put another brick in the wall by defining guidelines to help developers build specific usability characteristics into their applications. We have named these guidelines *usability guidelines for software development*.

We hypothesize that applying the proposed guidelines will

- reduce *development time*,
- reduce *perceived* usability-related functionality *complexity* for developers, and
- improve the *quality* of resulting software designs.

This paper details the empirical validation of these hypotheses while discussing the proposed guidelines for specific usability features, which we have termed functional usability features (usability features with major impact on software design). To do this, Section 2 explores related research. Section 3 describes the structure of our usability guidelines. Section 4 presents the guideline for the warning feature, and an application example is shown in Section 5. Section 6 details our empirical studies and results. Last, conclusions are outlined in Section 7.

- *L. Carvajal and A.M. Moreno are with the Facultad de Informática, Universidad Politécnica de Madrid, Campus de Montegancedo, Boadilla del Monte 28660, Madrid, Spain. E-mail: lauraelena.carvajal@upm.es, ammoreno@fi.upm.es.*
- *M.-I. Sánchez-Segura is with the Computer Science Department, Carlos III University of Madrid, Avenida de la Universidad 30, Leganes 28911, Madrid, Spain. E-mail: misanche@inf.uc3m.es.*
- *A. Seffah is with Concordia University, Montreal, QC, Canada. E-mail: seffah@encs.concordia.ca.*

Manuscript received 29 Nov. 2012; revised 4 Apr. 2013; accepted 26 May 2013; published online 3 June 2013.
Recommended for acceptance by J. Grundy.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2012-11-0347.
Digital Object Identifier no. 10.1109/TSE.2013.29.

TABLE 1
Functional Usability Features

Abort	Commands Aggregation
Undo	System Status Feedback
Multi-Level Help	Step by Step
Warning	Favorites
Progress Feedback	Preferences
Personal Object Space	

2 Motivation and Background

Over the past few years, some researchers have addressed the manner in which HCI recommendations can be integrated into mainstream software development. In [10], Bass and John developed one of the earliest approaches. They identified a set of usability scenarios and proposed architectural patterns to model the scenarios as solutions to particular usability problems. The patterns were expressed at a very high level of abstraction and later translated into more concrete solutions [12].

Bass et al. [12] supplemented their previous work by proposing actual class and sequence diagrams for each of the chosen usability scenarios. However, as explained in [9], only the cancel feature was ever fully fleshed out.

In [14], John et al. later shifted to a slightly different approach where they replaced the Unified Modeling Language (UML) diagrams with text-based recommendations for software architects as an integral part of their solutions. This decision was taken after testing their initial patterns in industry and discovering that developers seemed reluctant to accept ready-made UML and more readily warmed to text-only recommendations on how to design their systems.

In [15], Ferré et al. proposed possible architectural solutions for incorporating usability patterns into the architecture of software applications. They decomposed well-known usability attributes defined in the literature into more detailed attributes for which specific solutions could be proposed. These solutions were expressed in terms of architectural components, which were abstracted empirically by experimentation. However, they were still too abstract to be directly implementable by software developers.

In [16], Seffah et al. also identified usability scenarios. Instead of defining new solutions for each scenario, they proposed an algorithm for matching solutions to existing patterns. For example, the progress indicator pattern is proposed as a solution to the commonly known usability problem of time-consuming interactions that lead to user frustration and dissatisfaction.

While the results of this previous research have been promising, there are some limitations. The usability issues addressed were identified mostly using heuristic-based approaches rather than from the HCI literature. In addition, most existing research proposes highly abstract architectural solutions. While useful for depicting how a system should behave regarding usability issues, these solutions are difficult to transform into implementations. Finally, these research efforts do not provide any means of traceability between the proposed design solutions and software requirements.

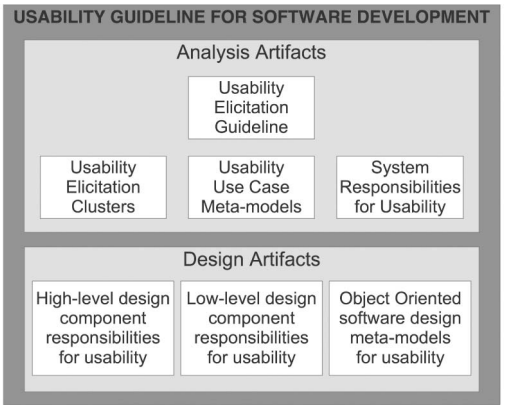

Fig. 1. Structure of the usability guidelines for software development.

Our work focuses on developing traceable lower-level design solutions within multipurpose guidelines that we have termed usability guidelines for software development. As we will see later, preliminary validation shows that these guidelines help developers build applications that include a set of usability features that are relevant from a HCI perspective and with a proven impact on software design.

3 Usability Guidelines for Software Development

We propose one guideline for each functional usability feature originally defined in [17] and shown in Table 1. Their relevance from an HCI perspective, as well as the reason why they should be dealt with as functional usability requirements, is discussed in [17].

As shown in Fig. 1, each guideline is comprised of seven artifacts; we refer to the first four as analysis artifacts and the last three as design artifacts. We have made this distinction for narrative purposes, but the boundary between analysis and design may shift from one project to another, and some projects may not even make a distinction between such activities. So, as we discuss later, developers can use these artifacts at their convenience, irrespective of their classification into the analysis or design categories. A brief description follows:

- Usability elicitation guideline (UEG) is an existing contribution by Juristo et al. [17] (extended for our purposes), designed to help analysts elicit usability requirements.
- Usability elicitation clusters are graphic representations of the UEG that help analysts understand the flow of the requirements discussion items.
- Usability use case metamodel is a use case representation of the usability needs covered by the UEG to help developers include such needs in their UC models.
- System responsibilities are the main functionalities that the system should accomplish to potentially fulfill all of what has been elicited using the UEG.
- High-level design component responsibilities for usability give an abstract description of the system responsibilities (software components).

TABLE 2
Usability 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 potentially serious consequences. Consequently, the application needs to warn users before executing the task so that if necessary, they can reconsider.			
Context			
Applications where user tasks may have 'potentially damaging' effects like loss of data			
Interrelationships			
Abort: Some warnings require a 'cancel' button. See 'cancel operation' section for the abort feature			
HCI Recommendation	**Explanation**	**Discussions w/Stakeholders**	**Usage Examples (optional)**
W_HCI-1 Warning types For each action that a user may take, consider the following aspects: its reversibility, its frequency, the degree of damage that it may cause and the immediacy of feedback in order to determine which type of warning needs to be given to the user: - Notification - Confirmation - Authentication	**W_ELAB-1** The more damage the action is likely to do, the higher the warning level should be. Be careful not to overdo it. Use notifications only when users benefit from the info provided. Warnings are preemptive, so error notification falls outside of the scope of this guideline (See Status Feedback).	**W_Q-1** Which actions require some type of warning? **W_Q-2** Which of these actions only require the user to be informed? **W_Q-3** Which of these actions prevent execution until the user gives confirmation? **W_Q-4** Which of these actions are highly damaging and need credential approval? **W_Q-5** For all actions, which information will be displayed for the user?	**W_EX-1 Notification** "Remember to…" messages. This action neither interrupts nor requires user feedback. **W_EX-2 Confirmation** "Are you sure that you want to…?" before executing damaging action. Users have to okay this action. **W_EX-3 Authentication** "You need to provide login and password before you can delete this file". Users have to provide credentials before the action can be executed.

- Low-level design component responsibilities for usability give a concrete description of system responsibilities in terms of classes, methods, and so on.
- Software design metamodels are the UML representation of the low-level design component responsibilities for usability.

Notice that our usability guidelines for software development differ from traditional usability and HCI guidelines. HCI guidelines list well-known principles for user interface design and evaluation [18]. For example, a general usability guideline could be "provide feedback to the user," which could be further specified as "ensure that the main objects of interest to the user are visible on the screen and that their most important attributes are shown" [4]. HCI patterns are one step beyond usability guidelines, providing details about when and why to use particular usability solutions, providing examples, and so on [19]. In any case, HCI recommendations focus on UI appearance and interaction, whereas our usability guidelines for software development focus on software analysis and design models. Our guidelines for software development complement traditional HCI guidelines. As we will see later, particular information from HCI guidelines has been used to define our guidelines for software development, specifically the usability elicitation guideline artifact.

4 Example: Warning Usability Guideline

The warning feature deals with the user's need to receive different alert types upon execution of sensitive (potentially damaging) actions. In this section, we briefly describe the usability guideline developed for this feature. For a more detailed description of this guideline, as well as of the guidelines for the other usability features listed in Table 1, see [20]. We discuss how to use this and the other guidelines in Section 5.

4.1 Analysis Artifacts

The first four artifacts in Fig. 1 are described below.

4.1.1 Usability Elicitation Guideline

Table 2 shows the UEG for warning. The first part of the guideline provides a general description of the feature. This is followed by more detailed information containing HCI recommendations that can condition analysis or design decisions, an explanation of these recommendations from a SE point of view, discussions to be had with stakeholders, and, finally, usage examples. As already mentioned, UEGs were discussed at length in [17].

4.1.2 Usability Elicitation Cluster Map

Fig. 2 represents the elicitation cluster map for the warning feature. It suggests the order in which the items recommended in the UEG shown in Fig. 1 should be discussed. A usability cluster map is primarily a flowchart representation of the UEG, helping analysts to determine the flow of the discussions and visually discard entire branches of discussion based on the stakeholder responses. Additionally, it groups these discussion items according to the topic that they cover. These groups (or clusters) result in the system responsibilities described in Section 4.1.4.

4.1.3 Use Case Metamodel

Fig. 3 shows the use case metamodel for warning. The applicability of these use cases will depend on the results of the elicitation process. If, for example, there is no need for

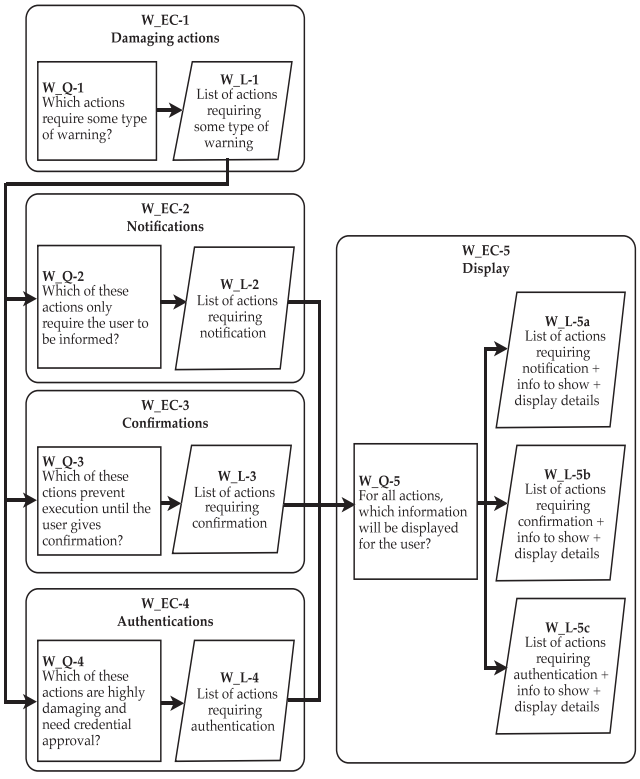

Fig. 2. Usability elicitation cluster map warning.

user authentication, use cases W_UC-3 and W_UC-6 would be discarded.

4.1.4 System Responsibilities

Table 3 shows the proposed system responsibilities for warning. As already mentioned, these are derived from the usability elicitation clusters.

Table 4 shows how responsibilities and clusters are related. A project that requires a specific cluster will also require its related system responsibilities. Likewise, if a cluster is discarded during elicitation, its related responsibilities will also be discarded.

4.2 Design Artifacts

The other three artifacts illustrated in Fig. 1 are described below.

4.2.1 High-Level Design Component Responsibilities

This artifact describes the system responsibilities for warning in terms of software components without alluding to any specific architecture, as shown in Table 5.

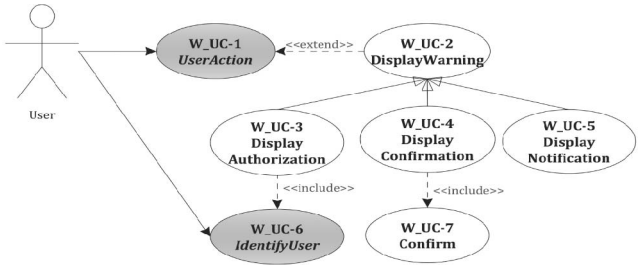

Fig. 3. Use case metamodel for warning.

TABLE 3
System Responsibilities for Usability: Warning

System Responsibilities List for Warning
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 actions require notifications
W-SR-3 Request confirmation
The system must be aware of which actions require user confirmation before execution
W-SR-4 Request authentication
The system must be aware of which actions require proper user authentication before execution
W-SR-5 Display warnings
The system must know which warning information to show for each action.

4.2.2 Low-Level Design Component Responsibilities

This artifact describes the system responsibilities in terms of concrete classes within a model-view-controller (MVC) architecture.

Table 6 shows the low-level design component responsibilities for warning as well as the sequence of actions involved in implementing this usability feature.

4.2.3 Usability Software Design Metamodels

These models represent the low-level design component responsibilities and are intended to be included in the final project UML design (white classes are to be copied directly, gray classes are substituted for the appropriate domain-specific class). Figs. 4 and 5 show the class and sequence diagrams for warning.

5 Usability Guidelines in Use

We now aim to provide a practical application of the usability guidelines for software development. So, we describe how they can be used with an example application: an online task manager. In particular, we discuss how to use the warning guideline presented in Section 4.

For clarity's sake, we follow the structure illustrated in Fig. 1 in this example, discussing analysis and design artifacts and applying the guidelines in full. However, as we discuss later, the guidelines can be adapted to different underlying software processes and developers may opt to use only the artifacts that are most relevant for their particular situation.

5.1 Elicitation and Analysis

During elicitation of this example project, the development team, typically the analyst, gathers a list of requirements. One of these requirements is shown below:

> Req(3) The system must allow users to delete any task from their task list.

The analyst will use the usability elicitation guideline and usability elicitation cluster map to add usability

TABLE 4
Clusters/Responsibilities Mapping: Warning

Elicitation Clusters	Dependent Responsibilities
W_EC-1 Damaging actions	W_SR-1 Be aware of damaging actions
W_EC-2 Notifications	W_SR-2 Notify
W_EC-3 Confirmations	W_SR-3 Request confirmation
W_EC-4 Authentications	W_SR-4 Request authentication
W_EC-5 Display	W_SR-5 Display warnings

TABLE 5
High-Level Design Component Responsibilities for Warning

System Resp.	High-Level Design Component Responsibilities
W_SR-1 Be aware of damaging actions	There must be a *wrapping component* for each *domain component* that includes at least one sensitive method. This component mimics the structure of the *domain component,* and it will 'sit' between it and any invoking class. All *wrapping components* must know whether it is safe to invoke actions in their *DomainComponent.* For example, if 'SalesItem' has a sensitive action, it must be renamed 'SalesItem*Domain*', for example, and this way its *wrapping component* can take the name 'SalesItem'. When called, actions in the *wrapping component* can respond that they *cannot yet* be executed, issuing a warning. This component is responsible for knowing which actions trigger which kind of warning and for triggering them when needed.
W_SR-2 Notify	If the issued *warning* is a *notification,* it must be sent to the *UI.* The call is then returned to the *wrapping component* for execution. As it is now safe, the *wrapping component* calls the action in the *DomainClass*.
W_SR-3 Request confirmation	If the issued *warning* is a *confirmation*, it must also be sent to the *UI component*. In this case, however, it must wait for the user to okay the invocation. After okaying, the invocation is returned to the *wrapping component* for execution. As it is now safe, the *wrapping component* invokes the action in the *DomainClass*.
W_SR-4 Request authentication	Finally, if the issued *warning* is an *authentication*, it must also be sent to the *UI component.* The *UI component* will then go through any necessary steps to cross-check credentials. After user authentication, the call is returned to the *wrapping component* for execution. As it is now safe, the *wrapping component* calls the action in the *DomainClass*.
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)

TABLE 6
Usability Guideline: Low-Level Design Component Responsibilities (MVC) for Warning

System Resp.	Objects				
	View	**Controller**	**DomainClassWrap**	***Domain Class***	**Warning**
W_SR-1 Be aware of damaging actions	1. *View* listens for and passes user calls (doAction())onto *Controller*	2. *Controller* is not aware of the existence of *DomainClassWraps*, it simply forwards the call to the (domain) class that it knows to be responsible for handling the method.	3. If a *DomainClassWrap* exists, it invokes its version of doAction() and checks if it can be safely executed. 4. If it is, it will forward the method call to *DomainClass*. 5b. Otherwise it will create the right type of warning.	5a. *Domain Class* executes the method.	
W_SR-2 Notify	2. When *View* receives a *Notification*, it displays its info. No user feedback is expected in this case, so control returns to the wrapper *via Controller.*	3. *Controller* requests the flag to be set for 'doAction' to OK. 4. It calls doAction() in *the wrapper* again, prompting a flag re-check.	5. Since the flag is now set to OK, the *DomainClassWrap* immediately forwards the method call to *DomainClass.*	6.*Domain Class* executes the method.	1. The new *Notification* is forwarded to *View.*
W_SR-3 Request confirmation	2. When *View* receives a *Confirmation*, it displays its info and the OK or Cancel option. If the user selects OK, control returns to the wrapper via *Controller*.	3. *Controller* requests the flag to be set for 'doAction' to OK. 4. It calls doAction() in *the wrapper* again, prompting a flag re-check.	5. Since the flag is now set to 'OK', the *DomainClassWrap* immediately forwards the method call to *DomainClass.*	6. *Domain Class* executes the method.	1. The new *Notification* is forwarded to *View.*
W_SR-4 Request authentication	2. When *View* receives an *Authentication*, it takes the appropriate actions to identify the user (involving other relevant domain classes). Once the user has been properly identified by the system, the control is returned to the wrapper via *Controller.*	3. *Controller* requests the flag to be set for 'doAction' to OK. 4. It calls doAction() in *the wrapper* again, prompting a flag re-check	5. Since the flag is now set to 'OK', the *DomainClassWrap* immediately forwards the method call to *DomainClass.*	6.*Domain Class* executes the method.	1. The new *Notification* is forwarded to *View*
W_SR-5 Display warning	*View* is responsible for displaying the info inside every *Warning* object.				

information to functional requirements like Req(3). For example, adding usability information about the warning feature would imply redefining the requirement as

> Req(3) The system must allow users to delete any task from their task list. *It must show an alert asking the user to confirm the action before permanently deleting the task.*

To do this, analysts must hold discussions with stakeholders according to recommendations by these two artifacts, supplementing the domain-specific requirements with the usability-related functionality.

In the case of Req(3), stakeholders suggest a list of actions that will require some sort of warning in response to W_Q-1, which user actions require the user to be warned in some way? Then, the analyst runs through questions W_Q2 to W_Q4 to decide which actions will require each of the three specific warning types: notification, confirmation, and

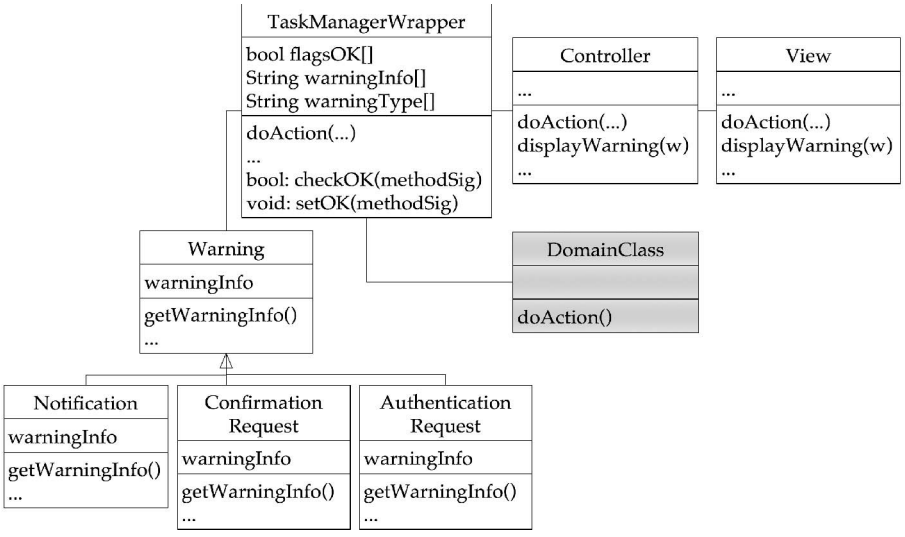

Fig. 4. Usability design metamodel. Warning class diagram.

authentication. In this example, only confirmation is required, and clusters 2 and 4 are discarded, as shown in Fig. 6. Once this has been established, discussions of W_Q-5 will determine the appearance of the actual confirmation and what information it will convey. This confirmation is likely to be a dialog box with OK and Cancel buttons and a message such as Are you sure you want to permanently delete this task? Depending on the project, the exact text of the warning message may be set at this point or left for later on in the development process. It should be clear at this point that some warning message is to be displayed.

The system use case model will contain use cases like the ones depicted in gray in Fig. 7. If this project also considers usability, it is supplemented with the remaining use cases (white) by applying the third artifact of our proposed guideline: the usability use case metamodels, as described in Section 4.1.3.

In this example, only confirmation is relevant to the requirements, and DisplayAuthentication, IdentifyUser, and DisplayNotification have been discarded. Accordingly, the development team would choose the relevant system responsibilities for usability for the warning feature, producing the subset of responsibilities shown in Table 7, as described in Section 4.1.4.

In this example, responsibilities W-SR-2 and W-SR-4 of the warning guideline (in Table 3) have been discarded, as

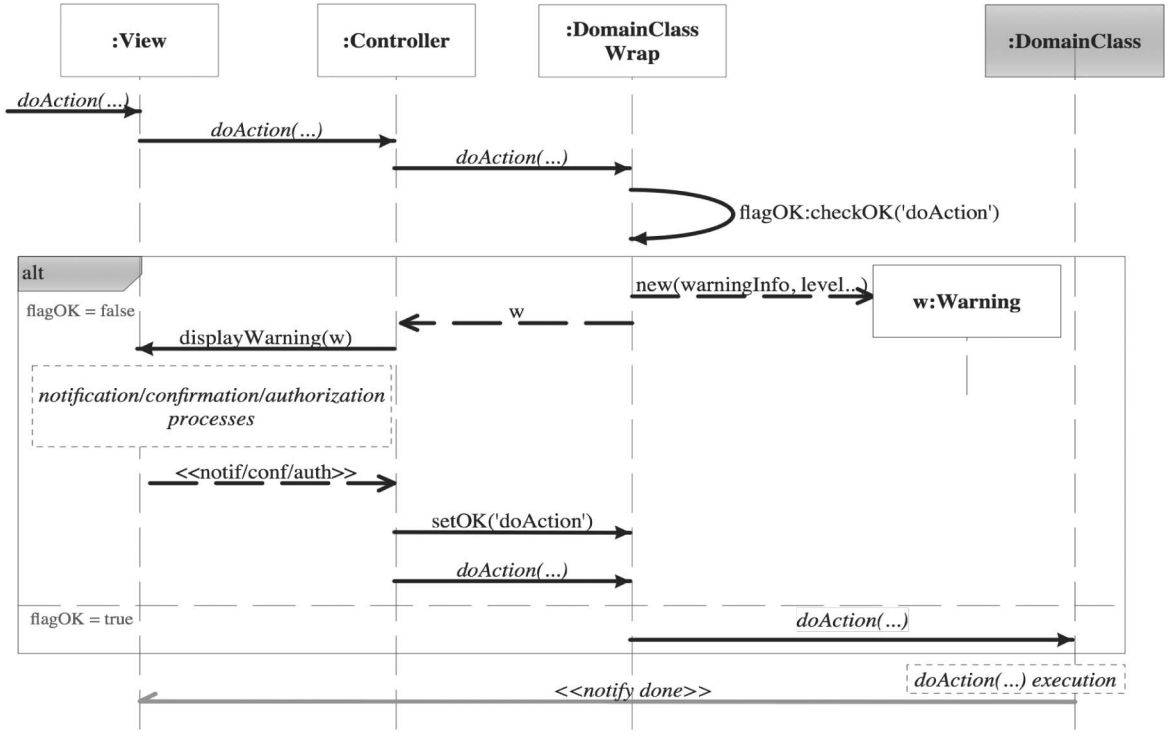

Fig. 5. Display warning sequence diagram.

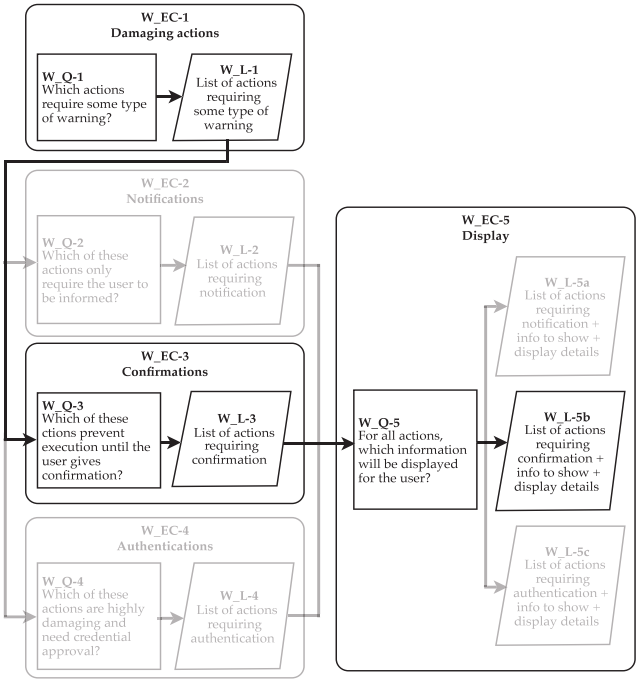

Fig. 6. Relevant elicitation clusters of warning feature.

the elicitation process determined that they would not be applicable for this project.

5.2 Design

During software design, the development team identifies the high- and/or low-level design component responsibilities that will be present in their system. To do so, they must discard responsibilities that correspond to the previously discarded system responsibilities and retain only the applicable component responsibilities shown in Table 7. The output is a subset of Tables 5 and 6, where rows W-SR-1, W-SR-2, and W-SR-4 are eliminated. These two tables determine which software components should be present in the system to fulfill the applicable system responsibilities, and what their responsibilities need to be from both a more abstract (high-level) and MVC-specific (low-level) perspective. Non-MVC projects may choose to keep only the high-level responsibilities and to stop using the guideline at this point, whereas MVC projects may skip the high-level responsibilities and directly produce the list of low-level responsibilities. These responsibilities

TABLE 7
Applicable Subset of System Responsibilities for Warning in Example Project

W-SR-1 Be aware of damaging actions
The system must know which actions will require warnings
W-SR-3 Request confirmation
The system must be aware of which actions require user confirmation before execution
W-SR-5 Display warning
The system must know which warning information to show for each action.

express the interaction of software components for an MVC architecture.

The development team can use the usability software design metamodels with the low-level software component responsibilities to output the class and interaction diagrams related to the corresponding feature.

A partial view of the class diagram not considering usability for this example might be made up of the domain classes shown in gray in Fig. 8. These are enhanced with the remaining usability-related classes for the warning feature (white) when applying the usability software design metamodels as described in Section 4.2.3.

Similarly, the sequence diagram for the functionality studied in this example (deleting a task) is supplemented with usability according to the sequence diagram of the usability software design metamodel for warning in Fig. 6. The resulting sequence diagram is shown in Fig. 9.

This section has shown a small example of the application of our guidelines. The procedure depicted here should be used to include other usability features applicable to the online task manager example (undo and progress). Fig. 10 shows a simplified version of the full class diagram for this example. This diagram includes the domain-specific classes (gray) as well as the usability-related classes (white) of the applicable usability features. More details on this system, the full set of sequence diagrams, and remaining products are available in [20].

Clearly, the design artifacts provided in the guidelines are designed to develop usability functionalities wrapped as classes that manage the respective facility. Additionally, the set of classes that constitutes a usability solution for each feature is loosely coupled with the remainder of the system. Accordingly, all the functionalities affected by the usability feature can share the usability classes, and there is

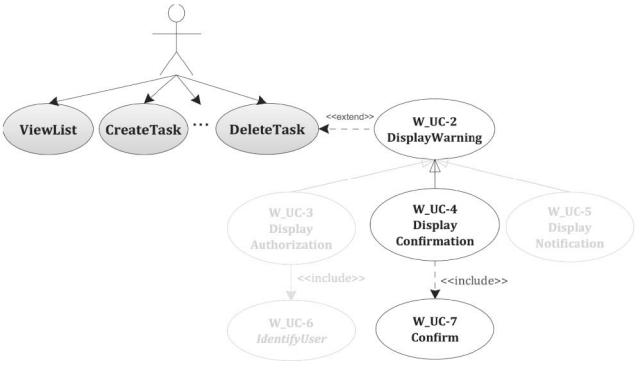

Fig. 7. Example use case model including usability (warning feature).

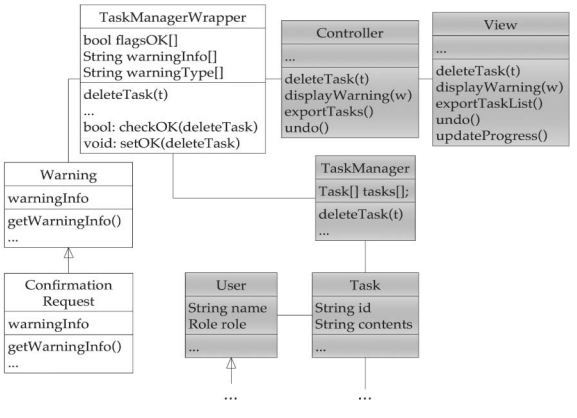

Fig. 8. Example class diagram including usability (warning feature).

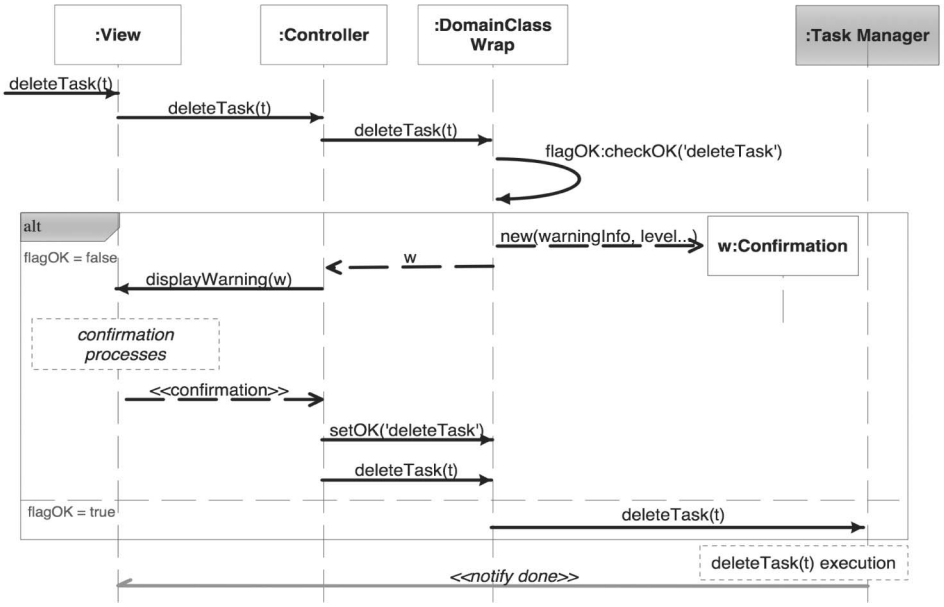

Fig. 9. Example sequence diagram including usability (warning feature).

not a huge increase in additional usability classes when the applications are scaled up.

One limitation of these guidelines is that they can be impractical to manage on paper. This applies especially to UML diagrams, where developers would have to copy classes and methods, one by one, into their UML design tools. On this ground, we have developed an automated tool which is beyond the scope of this paper, but is described in detail in [20]. Developers using this tool have the option to produce their instantiated UML diagrams for all usability features in .xmi format. It then takes developers just a few clicks to import these files into an UML design tool of their choice and build them into their domain-specific designs, thus circumventing the need to manually copy UML elements.

Finally, notice that our guidelines provide recommendations to incorporate particular usability features into the software analysis and design processes, that is, they do not focus on the look and feel of these features in the user interface. As already mentioned, usability guidelines and patterns for this purpose have already been reported in the literature. For example, HCI patterns provide solutions that can complement our guidelines with recommendations about how the respective usability features should be presented to users.

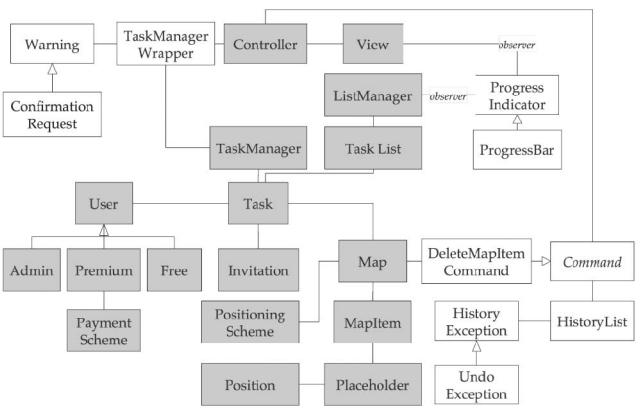

Fig. 10. Simplified class diagram for a full example system.

6 Empirical Study

Our proposed guidelines were applied across several projects to test the research hypotheses. As already mentioned, the usability guidelines for software development can be applied fully or partially. Our aim was to compare not only the effectiveness of applying versus not applying the guidelines, but also to gauge the contribution of specific design artifacts. So, for validation purposes, we chose to compare three ways of using the guidelines: full guidelines or FG (applying all artifacts), partial guidelines or PG (applying analysis artifacts only), and no guidelines or NG.

6.1 Hypotheses

We aimed to test the hypotheses stated in the introduction, according to which applying the proposed usability guidelines for software development (as opposed to not applying or applying only part of) will

- reduce the *development time*,
- reduce the *perceived* usability-related functionality *complexity* for developers, and
- improve the *quality* of resulting *software designs*.

As there are three possible modes of use (NG, PG, and FG), the research hypotheses are refined for validation purposes as follows:

- *H0-1*. There are no significant differences in mean development time among the three modes of use.
- *H-1*. There are significant differences in mean development time among the three modes.
- *H0-2*. There are no significant differences in mean perceived complexity among the three modes.
- *H-2*. There are significant differences in mean perceived complexity among the three modes.

TABLE 8
Perceived Complexity Questionnaire for Undo

Please rate how hard you found it to design (a) and implement (b) the Undo feature. Use the 1-to-5 scale provided where "1" means "very easy" and "5" means "very difficult"
Ease of Design (1 2 3 4 5) Rate the ease of design of the Undo feature. Consider how long it took you to fully understand the goal of this feature within your system, how many times you had to iterate over to achieve that goal, how this feature was to interact with other functionality and whether or not you had to redesign the feature after implementation
Ease of Implementation (1 2 3 4 5) Rate the ease of implementation of the Undo feature. Consider how straightforward it was to write the code for this feature, whether any refactoring was needed, the complexity of the classes involved and whether or not you had to re-write your code after testing.

- *H0-3*. There are no significant differences in mean design quality among the three modes.
- *H-3*. There are significant differences in mean design quality among the three modes.

For each set of hypotheses, if significant differences in means are found among the three groups, they are further analyzed to determine exactly which pairs are causing the difference (FG-NG, FG-PG or PG-NG).

We now describe how we tested these hypotheses, the observed variables, the participants, and the data collected. Finally, we analyze these data and report the results and findings of the validation process.

6.2 Experimental Design

6.2.1 Variables

Our independent variable (or experimental factor) is the guidelines' mode of use, with three alternatives (NG, PG, and FG). In this way, participants using NG were asked to develop each project without assistance for incorporating usability issues. Participants using PG were only given the analysis artifacts of the guideline (up to the system responsibilities) and received no instructions on how to go about design. Participants using FG were asked to develop each project applying the analysis and design artifacts of the guidelines.

As for dependent variables, the following were tracked by in each project: development time, perceived complexity, and design quality. Development time was measured by participants for each phase (analysis, design, implementation, and testing) in minutes (ratio variable) and then summed up to get the total development time.

Perceived complexity was scored by participants on a 1-to-5 scale to rate how hard they found each usability feature to design and implement (interval variable). Perceived complexity is a subjective variable. However, the aim was for all participants to have similar criteria in mind when assessing this variable. Therefore, they were asked to rate this perception for each usability feature according to a similar set of instructions. Table 8 shows them for the Undo feature.

Design quality (interval variable) was defined as a combination of the following quality attributes: notation correctness, adequate responsibility allocation, and diagram readability [1], [20]. An experienced UPM object oriented (OO) design professor rated these quality attributes for each project on a 1-to-5 scale ($1 =$ lower quality, $5 =$ higher quality). This professor was uninvolved with either constructing the guidelines or running the experiment.

Notice that we did not address UI metrics, as our guidelines do not focus on the UI. Our aim was to check how such guidelines can help developers to deal with usability during software analysis and design. As we will see later, participants dealt with UI issues as best they could; they were not provided HCI guidelines or patterns for recommendations about how to present the respective usability features.

6.2.2 Experimental Units

In this experiment, participants worked on three similar projects to assure that the experimental results were not project dependent (projects are available from http://is.ls.fi.upm.es/miembros/amoreno/srs). The projects used were as follows:

1. *An online task manager*. An application to manage to-do lists, sharing and scheduling, as well as visually organizing tasks.
2. *A console for a home automation system*. An application to operate a simulated network of sensors and actuators that control various features of a home environment in real time.
3. *An auction site*. A web application with basic auction functionalities.

These are academic projects, and therefore their complexity and size is limited (to 250 function points each). Experiments with academic projects are common in the SE literature as initial sources of evidence (see, for example, [21], [22]). This evidence needs to be further validated in other academic and industrial, settings.

6.2.3 Participants

We worked with nine 2010-2011 SE master's degree students at UPM's School of Computing (experiments with a small number of participants are also common in the SE literature due to the characteristics of SE experiments; see, for example, [23]). The SE Master at UPM is a one-year program. Students are admitted to this program after passing a strict selection procedure based on their average academic outcome during their bachelor's degree work in computing. Accordingly, program students form a homogeneous student group with a high undergraduate attainment level. They participated in the experiment as part of the project that they had to develop during an SE Master course unit: software project. All nine students who participated in the experiment had taken the same master's course units (none of which were related to usability) and had similar grade point averages (GPA) ranging from 8 to 9.5 on a 0-to-10 scale, as can be seen in Table 9 (mean value 8.72 and s.d. 0.56). They could also be considered as junior software practitioners with between 6 and 12 months of work experience in industry.

Therefore, even though the number of participants in this experiment is small, they did constitute a homogeneous

TABLE 9
Participants GPA and Experience

	GPA	Ind. Exp. (months)
Participant 1	8.40	11
Participant 2	8.90	9
Participant 3	8.90	7
Participant 4	8.30	6
Participant 5	9.00	9
Participant 6	8.00	8
Participant 7	9.30	6
Participant 8	9.20	12
Participant 9	8.50	12
Mean	8.722	8,81
Stdv.	0.441	1,67

group whereby it is feasible to apply randomization to allocate participants. Thus, each participant was randomly assigned to a mode of use, as shown in Table 10. In sum, each alternative of the independent variable (mode of use) is internally replicated three times using three experimental units (participants were also randomly allocated to experimental units).

6.3 Experimental Procedure

As already mentioned, the experiment was run in the context of the software project course unit. Instructors met each student individually to explain the project that they were going to develop. They were not told at any point that the project was part of an experiment. Students using FG were given a short, approximately 45-minute, tutorial about the analysis and design artifacts of the guidelines. Students using PG received a short 15-minute tutorial on the analysis artifacts.

For each project, students were given a software requirements specification document (SRS) containing the respective functional requirements, as well as the usability requirements related to the features listed in Table 1. So, in this case, participants were relieved of the task of eliciting requirements including usability ones, as the effectiveness of the usability elicitation guidelines has already been proven in [17].

All participants were asked to develop their projects iteratively, covering analysis, design, implementation, and testing in each iteration. They were asked to produce analysis artifacts, such as use case models, and design artifacts, such as class and collaboration diagrams. All groups used the free ArgoUML tool to draw UML design diagrams (the guidelines were applied manually by the PG and FG groups). None of the participants used the automated tool mentioned in Section 5.2. This policy was meant to prevent the FG and PG groups from having a potentially unfair advantage over NG groups.

Participants tested their software code at the end of each iteration by applying JUnit to run automated unit tests. Any detected defects were fixed, and the automated tests were rerun for the purposes of confirmation.

TABLE 10
Participants by Project and Mode of Use

	Independent Variable		
Experimental Unit	**NG**	**PG**	**FG**
Task Manager	Participant 3	Participant 1	Participant 5
Home Automation	Participant 7	Participant 4	Participant 6
Auction Site	Participant 2	Participant 9	Participant 8

TABLE 11
Average Total Time (in Min)

	NG		PG		FG	
	Average	s.d.	Average	s.d.	Average	s.d.
Analysis	250.00	*91.8*	215.33	*26.5*	247.33	*73.7*
Design	536.00	*132.9*	365.67	*79.1*	207.75	*14*
Impl.	771.67	*580.3*	579.67	*75.2*	334.71	*158.8*
Testing	408.67	*127.2*	206.67	*100*	23.67	*9.3*
TOTAL	1,966.33		1,367.33		813.46	

Participants were given a template to record the time data for each activity. They were asked to record partial data continuously (that is, they were expected to record how much time they had spent on each particular phase at the end of each day). The times were added up at the end of each phase. This was designed to assure that time data were as reliable as possible.

Participants were told that the course assessment criteria would focus on the quality of the resulting designs (rated according to the variables described in Section 6.2.1), irrespective of the recorded time data or their perception about the complexity of the usability features. Participants were asked to develop the UI to the best of their ability. They were not provided with any information about HCI or UI patterns or guidelines, as our aim was related to the software design aspects of the projects. Participants were asked to just provide a working interface, and they were told that their projects would not be assessed on UI appearance.

Instructors met with each participant on a fortnightly basis to check project progress. Instructors did not assess the project at these meetings; they merely cleared up possible misunderstandings concerning the functional requirements and helped participants plan their work for the next meeting. Each project was considered finished only after it was fully developed and functional in terms of both expected system functionality and the usability features listed in Table 1. Projects had to be locally deployed by participants and used satisfactorily, hands-on, by instructors.

6.4 Data

This section outlines the data collected for each variable during experimentation.

6.4.1 Development Time Data

Table 11 shows the average time spent by participants on developing their projects.

The above values were obtained by averaging and adding up the times taken by participants (during each of these phases). The sharp differences in testing time are due to the fact that, as discussed earlier, testing was performed automatically and participants who found few or no defects completed this phase very quickly. More importantly though, testing time also measured the time it took to fix these defects. Participants who found dozens of defects also had to measure the time it took to fix the errors, further increasing testing time.

6.4.2 Perceived Complexity Data

Table 12 shows the mean responses to questionnaires administered to participants who were asked to score

TABLE 12
Average Perceived Complexity of Functional Usability Features as Reported by Participants

	NG		PG		FG	
	Average	s.d.	Average	s.d.	Average	s.d.
Design	3.32	1.08	2.70	*1.08*	1.73	*0.94*
Implementation	3.33	1.08	2.62	*1.12*	1.76	*0.90*
Avg	**3.33**	**1.07**	**2.66**	***1.09***	**1.74**	***0.91***

perceived usability feature development complexity. The first two rows show the average ranking by participants of how hard they found the usability features to design and implement, respectively, on a 1-to-5 scale. The columns denote the guideline type (NG, PG, and FG) that was used.

The data in Table 12 were obtained by averaging the participant's scores of all functional usability features.

6.4.3 Design Quality Data

Table 13 shows the mean scores for the design quality data of the three groups of participants: NG, PG, and FG. These data were obtained by averaging the scores for the designs by each participant.

6.5 Analysis

In view of the characteristics and size of the experimental sample, we used the Kruskal-Wallis (KW) test. This test can be applied with small sample sizes (see [24] for information about the application of the test to identify differences between three groups of three elements, as is our study, and the critical values used in these cases). The KW test assumes that sample populations should be identically shaped and scaled (which is also our case because, as discussed previously, the sample population comes from a group of homogeneous students). To test our assumption that project type does not have a significant effect on this experiment, we performed KW tests on the response variables grouped by project and found no statistical differences among them. The p-values obtained were always greater than 0.05 (0.670, 0.556, and 0.965, for time, complexity, and quality, respectively). Accordingly, we applied KW tests to the three alternatives (NG, PG, and FG) for each variable globally and by development phase, using a significance level of 0.05 and a confidence interval of 95 percent. Further (adjusted) pairwise testing was conducted for every case in which the KW result was significant to determine exactly which alternatives were different. Additionally, Tamhane tests were conducted in cases where KW tests resulted in p-values that were overly close to the threshold. We used SPGS Statistics version 2.0 for the statistical calculations.

The following sections show the analysis of the three dependent variables.

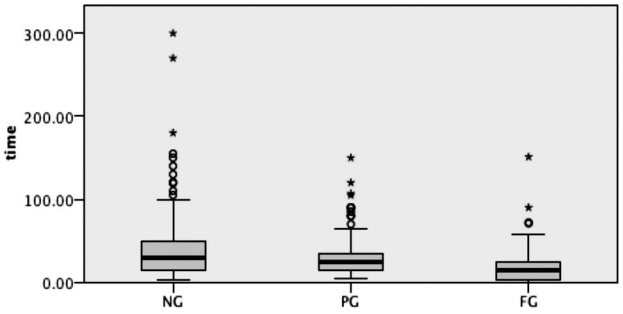

Fig. 11. KW test results for total development.

6.5.1 Development Time

There follows an analysis of the development time data and an analysis for the same data over the four development phases individually: analysis, design, implementation, and testing. The hypotheses of our experiment refer not to the development time for each separate phase but to the total time taken, calculated as the sum of the times for each phase. However, as we have access to the phase times, we think it is worthwhile to analyze tham along with global development time. This helps us to better identify in which parts of the development process time differences appear.

Global development time. As shown in Table 11, FG participants took on average less time than PG participants, who again took less time than their NG counterparts, to develop their projects. The KW test shows that the means differ across the groups with a significance of $p < 0.05$, as illustrated in Fig. 11.

To determine exactly which groups were different from each other, we ran a KW test for pairwise comparisons. Table 14 shows the results: The differences in means are statistically significant for FG-NG and FG-PG, but not for PG-NG.

Development time by phase. The next step in the analysis is to determine where the development time differences exist not only globally but also within each development phase. Fig. 12 shows the time data from Table 9 discriminated by phase.

For all but the analysis phase, participants who used FG took on average less time than their NG and PG counterparts. Four separate KW tests were performed on these data to confirm this observation. Table 15 presents the resulting p-values.

These results show no evidence that the three means (NG, PG, and FG) are statistically different for the analysis phase. For all other development phases, the test shows that the means are different across groups. To determine exactly which groups are different in each case, the KW test was executed for pairwise comparisons for the

TABLE 13
Average Quality Score for Resulting Designs

	NG		PG		FG	
	Average	s.d.	Average	s.d.	Average	s.d.
Adequate R. Allocation	2.33	*0.58*	2.33	*0.58*	5.00	*0.00*
Diagram Readability	3.33	*0.58*	2.67	*1.15*	4.67	*0.58*
Notation Correctness	4.00	*0.00*	3.67	*0.58*	4.67	*0.58*
Average quality score	**3.22**	***0.83***	**2.89**	***0.93***	**4.78**	***0.44***

TABLE 14
Pairwise KW Test Results for Total Development Time of Usability Features

	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

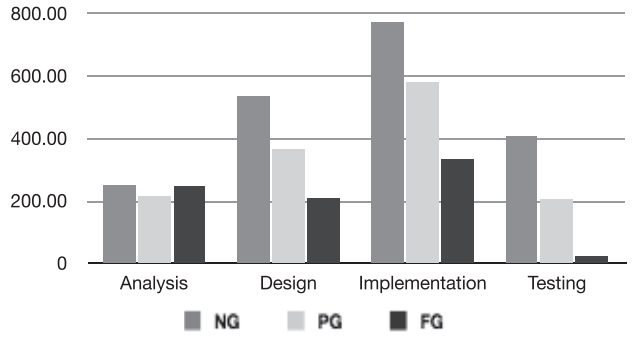

Fig. 12. Average time (min) to develop all usability-related functionality by development phase.

TABLE 15
KW Test Results for Total Development Time of Usability Features by Phase

	analysis	design	implementation	testing
p-value	0.778	0.000*	0.006*	0.000*

TABLE 16
Pairwise KW Test Results for Total Development Time of Usability Features by Phase

	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj. Sig.
DESIGN PHASE					
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
IMPLEMENTATION PHASE					
	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj. Sig.
FG-PG	17.788	7.062	2.519	.012	**.035***
FG-NG	21.076	7.062	2.984	.003	**.000***
PG-NG	-3.288	7.062	-.466	.642	1.000
TESTING PHASE					
	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	.190	.057

development phases that proved different. The results are shown in Table 16.

For the design phase, we find that the FG-PG (i.e., developers using the partial guidelines took longer to complete their design than developers using the full guidelines), and FG-NG (developers who did not apply the guidelines took longer to complete their design than developers using the full guidelines) groups are different. Furthermore, no statistically significant difference in means was found for PG-NG.

The same applied for implementation and testing: There were statistically significant differences in means between FG and PG, and also between FG and NG, but no evidence was found for the case of PG-NG.

Additionally, a Tamhane test was performed on these same data and confirmed the results of these four KW tests. This test further showed a statistically significant difference between the PG and NG pair for the testing phase, which had scored only slightly above the confidence threshold for the KW test.

6.5.2 Perceived Complexity

There follows an analysis of the perceived complexity data for entire project development, as well as for the individual

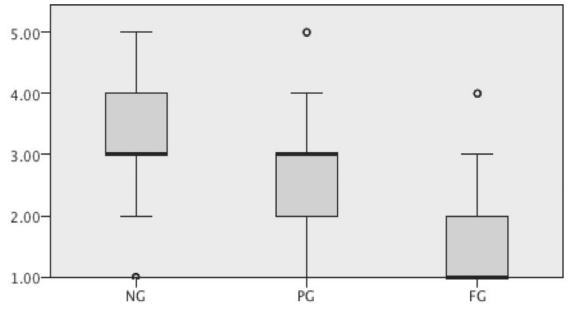

Fig. 13. KW test results for overall perceived complexity.

TABLE 17
Pairwise KW Test Results for Overall Perceived Complexity of Usability Features

	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***

project phases. Cronbach's alpha test was applied to test the internal consistency of the data gathered for the complexity variable. First, we applied the test to the data assessing the complexity of design and the complexity of implementation separately, which yielded alpha values of 0.816 and 0.842, respectively. Once we knew that there exists consistency within each subvariable, we applied the test to check the consistency between them resulted in an alpha value of 0.968. These data confirm that the perceived complexity can be studied by both aggregating the data for each individual project phase as well as on a phase-by-phase basis.

Global complexity. As shown in Table 12, FG participants found the usability features less complex than PG participants, who again found them less complex than NG participants. A KW test confirms that means differ across the three groups significantly. Fig. 13 illustrates these results.

Pairwise KW tests showed that the differences in means are statistically significant between all three pairs. These results are shown in Table 17.

Complexity by phase. The next step in the analysis is to determine whether such differences in terms of perceived complexity exist not only globally but also during design and implementation. Fig. 14 shows the data from Table 12, where FG users perceived guidelines as being less complex than NG or PG users did during design and implementation.

Two KW tests were performed on these averages, one for each group, confirming that these differences in average perception of complexity are statistically significant, as shown by the p-values in Table 18.

To determine which of the three groups are different in both cases, the KW test was executed for pairwise comparisons.

For both design and implementation, Table 19 shows that the groups that are different are FG versus PG and FG versus NG. Furthermore, no statistically significant difference in means was found for PG versus NG.

Additionally, a Tamhane test was run on these same data and found that there was a significant difference between the PG-NG pair, which had scored only slightly above the confidence threshold for KW.

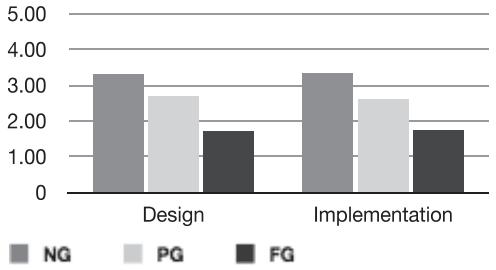

Fig. 14. Average perceived complexity of all features by phase.

TABLE 18
KW Test Results for Overall Perceived Complexity of Usability Features by Phase

	Design	Implementation
p-value	**0.000***	**0.000***

TABLE 19
Pairwise KW Test Results for Overall Perceived Complexity of Usability Features by Phase

DESIGN PHASE					
	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
IMPLEMENTATION PHASE					
	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj. Sig.
FG-PG	30.348	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

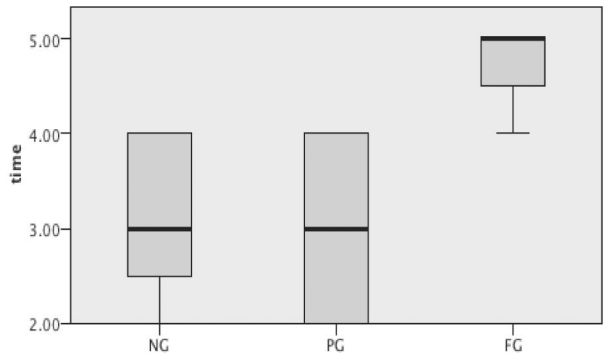

Fig. 15. KW test results for overall quality scores.

6.5.3 Design Quality

There follows an analysis for the design quality data for entire project development, followed by an analysis of these time data for each individual project phase.

Global quality. As shown earlier in Table 13, the designs of FG participants scored higher on average than those of the PG and NG participants.

A KW test determined that these three means are significantly different from one another with a confidence level of over 95 percent. Fig. 15 illustrates these results.

To determine which of the groups were different from each other, the KW test was applied for pairwise comparisons. Table 20 shows that the means are statistically different for NG versus FG and PG versus FG, but not for the PG-NG pair.

Analysis for the quality variable by attribute. The next step in the analysis is to determine if such differences in terms of design quality exist not only overall but also within each attribute. Fig. 16 shows the average scores for the three selected quality attributes.

TABLE 20
Pairwise KW Test Results for Quality Scores of Usability Features

	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj. Sig.
FG-PG	-13.333	3.607	-3.696	.000	**.001***
FG-NG	-11.333	3.607	-3.142	.002	**.005***
PG-NG	2.000	3.607	.554	.579	1.000

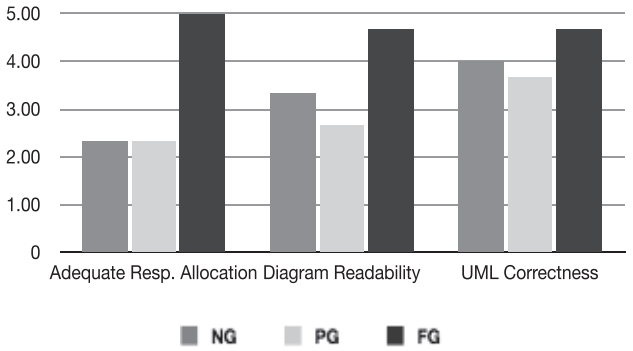

Fig. 16. Average scores for the designs for quality attributes.

TABLE 21
KW Test Results for Quality Scores by Quality Attribute

	Adequate Responsibility Allocation	Diagram Readability	Notation Correctness (UML)
p-value	**0.046***	**0.059†**	**0.086†**

TABLE 22
KW Test Results for Adequate Responsibility Allocation Quality Attribute

	Test Statistic	Std. Error	Std. Test Statistic	Sig.	Adj. Sig.
FG-PG	-4.500	2.092	-2.151	.031	.094
FG-NG	-4.500	2.092	-2.151	.031	.094
PG-NG	.000	2.092	.000	1.000	1.000

The resulting designs of participants who used the FG scored higher than their NG and PG counterparts across the board. Furthermore, the resulting designs of NG and PG participants were similar in quality in all cases, where designs by NG participants had a slight edge for diagram readability and notation correctness.

Three KW tests were performed, one for each attribute, to determine the significance of these mean differences. Table 21 shows the results.

These results suggest that for the adequate responsibility allocation attribute there is evidence to claim that the differences in means are indeed significant among the groups. For diagram readability and notation correctness, the means cannot be proven to be different at the desired confidence level of 95 percent, though they are at a 90 percent confidence level (represented by †).

For the adequate responsibility allocation attribute, adjusted pairwise comparisons show no difference between any of the pairs, as shown in Table 22.

Though seemingly contradictory, given that the three-way test indicates the existence of a within-group difference in means, 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

(which were 0.031 before adjustment for our FG-PG and FG-NP pairs). Since the first test turned out to be significant, it was worth confirming its source, albeit with an alternative test. For this purpose, we ran a Tamhane test with the same confidence interval on these data. This test indicated that differences are indeed caused by the FG-NG and FG-PG pairs. This test did not return any evidence of difference in means for the PG-NG pair.

6.6 Results and Findings

Our research hypotheses state that using the proposed usability guidelines for software development helps reduce development time, improves design quality, and reduces perceived complexity of usability features.

Regarding time, results showed that participants who applied the full guidelines developed their projects more quickly than developers who did not apply these guidelines, leading us to reject our first null hypothesis, H0-1.

Specifically, participants who applied the full guidelines created their designs more quickly than developers who applied partial or no guidelines. This leads us to believe that applying the proposed guidelines helped them form a clearer idea about what their designs would look like sooner than developers who did not use the guidelines. The group that did not use guidelines had to design their systems from scratch and had to do more design iterations before they were confident enough to implement them. Better designs led to smoother implementations. They also led to much quicker testing, considering that groups using the full guidelines not only found very few defects during testing (and certainly fewer than the other two groups) but also spent less development time on fixing these defects in their application than the other two groups. The extra development time was measured as part of the testing phase, thus making the differences between groups even more noticeable in this phase (see Fig. 12). Furthermore, to fix defects, these two groups not only had to change (and retest) software code but sometimes also had to change their original design, thereby further increasing the total time spent on the testing phase.

Regarding the perceived complexity of usability features, participants who applied the full guidelines generally found the usability features easier to develop than participants applying the partial guidelines and even more so than their peers using no guidelines. This led us to reject our H0-2 null hypothesis.

The quality of the resulting designs was generally higher for participants who applied the full guidelines than for developers who did not use guidelines or used partial guidelines, leading us to reject the second null hypothesis, H0-3.

Participants who applied the full guidelines produced designs that scored, on average, significantly higher for adequate responsibility allocation than the other two groups. Regarding diagram readability and notation correctness, they also improved in projects developed with the full guidelines. So, from a practical point of view, this means that the guideline design artifacts help developers to create quality designs.

Notice that more significant improvements were found for participants using the full guidelines versus the partial guidelines and no guidelines, and less clear differences were found for participants using the partial versus no guidelines. This led us to identify the added value of the design artifacts included in the full guideline for novice practitioners using MVC and OO models.

6.7 Threats to Validity

While these results are highly encouraging, they may not be fully generalizable because participants had low industrial expertise and the projects used, although real, were developed in an academic context and had to be of a reasonable size for completion in the available time. Furthermore, each project domain could potentially represent a moderating factor, although working with different domains reduced this threat. Having in mind that the number of participants is not high, possible differences among them could also affect the results of the experiment. To manage this issue, we worked with a homogeneous student group whose members had a high undergraduate attainment level and very similar and high grades in their master course units. Any communication among the participants during the projects development could also have an impact on the results. However, in view of the between-group differences in results, no such communication is likely to have taken place.

The development time variable was measured by each participant. As already mentioned, we addressed this threat by explicitly asking participants to record this variable from the start of the development process and giving them a template that they were to use to record the data. Neither were they informed of the purpose of the experiment, removing any possibility of reporting bias. We also gave participants reassurance that the time data they reported would not be used for grading purposes.

Another possible source of bias is related to the grading of the quality variables. To manage this issue, one reviewer reviewed all projects, blind to which project had been developed according to each mode of use.

7 Conclusion

As highlighted in this paper, incorporating usability features with a high impact on software logic is not a straightforward task. Some relevant research has been developed in this direction, but this is still an open question.

In this paper, we set out to contribute to this field by proposing usability guidelines for software development describing a possible solution for incorporating some of the best-known usability features into software applications. The key guideline artifacts specify the responsibilities that the system and its parts must fulfill to conform to these usability features, making them directly implementable from design.

Preliminary validation results show a significant decrease in development time; developers who applied the guidelines built their designs more quickly. Furthermore, applying the guidelines also reduced testing time. Applying the guidelines also helped developers produce enhanced designs, especially designs with better responsibility allocation. Finally, the proposed guidelines helped developers perceive usability features as not being overly complex. This finding is crucial as this perception could ultimately predispose a development team toward or against usability.

Although the proposed guidelines are composed of seven artifacts, they should be tailored to each project. For

example, if the team has usability experts who provide usability requirements, developers can skip the elicitation guidelines and clusters and use the system responsibilities and subsequent artifacts; developers who are using MVC but not developing explicit OO models can use the low-level design components discarding the OO design meta-models; and developers who are not using an MVC architecture can stop using the guideline after identifying the high-level responsibilities. Notice also that the proposed guidelines constitute one possible solution for incorporating the respective usability features into a software application.

Notice also that software engineers must tradeoff usability against the other quality attributes (performance, security, ...) in each software product. This tradeoff should be carefully studied according to the final quality required by the product.

We are in the process of testing our guidelines in an industrial setting. To do this, practitioners will work with the tool that we have mentioned to lighten the workload of using the guidelines on paper.

REFERENCES

[1] ISO IEC 9126-1:2001, "Software Engineering-Product Quality-Part 1-Quality Model," 2001.

[2] K. Nebe and V. Paelke, "Usability-Engineering-Requirements as a Basis for the Integration with Software Engineering," *Proc. 13th Int'l Conf. Human-Computer Interaction, Part I: New Trends,* pp. 652-659, 2009.

[3] R.M. Bias, *Cost-Justifying Usability: An Update for the Internet Age.* Elsevier, 2005.

[4] J. Nielsen, *Usability Engineering.* Morgan Kaufmann, 1993.

[5] J. Nielsen, "Usability ROI Declining, but Still Strong," *Alertbox.* http://www.useit.com/alertbox/roi.html, Jan. 2008.

[6] A. Seffah, "The Obstacles and Myths of Usability and Software Engineering," *Comm. ACM,* vol. 47, no. 12, pp. 71-76, 2004.

[7] S. Kujala, "Linking User Needs and Use Case-Driven Requirements Engineering," *Human-Centered Software Eng.—Integrating Usability Software Development Lifecycle,* A. Seffah, J. Gulliksen, and M.C. Desmarais, eds., Springer, 2005.

[8] H. Fischer and F. Nebe, "A Holistic Model for Integrating Usability Engineering and Software Engineering Enriched with Marketing Activities," *Proc. Second Int'l Conf. Human Centered Design,* pp. 28-37, 2011.

[9] L. Bass, B.E. John, and J. Kates, "Achieving Usability through Software Architecture," Technical Report CMU/SEI-2001-TR-005, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, Penn., Mar. 2001.

[10] L. Bass and B. John, "Linking Usability to Software Architecture Patterns through General Scenarios," *J. Systems and Software,* vol. 66, no. 3, pp. 188-197, 2002.

[11] E. Folmer, J. van Gurp, and J. Bosch, "Software Architecture Analysis of Usability," *Proc. Int'l Conf. Eng. Human Computer Interaction and Interactive Systems,* pp. 38-58, 2005.

[12] B. John, L. Bass, and M. Sanchez-Segura, "Bringing Usability Concerns to the Design of Software Architecture," *Proc. Int'l Conf. Eng. Human Computer Interaction and Interactive Systems,* pp. 1-19, 2005.

[13] N. Juristo, A.M. Moreno, and M. Sanchez-Segura, "Analysing the Impact of Usability on Software Design," *J. Systems and Software,* vol. 80, no. 9, pp. 1507-1516, 2007.

[14] B. John, L. Bass, E. Golden, and P. Stoll, "A Responsibility-Based Pattern Language for Usability-Supporting Architectural Patterns," *Proc. First ACM SIGCHI Symp. Eng. Interactive Computing Systems,* 2009.

[15] X. Ferre, N. Juristo, A.M. Moreno, and M. Sánchez-Segura, "A Software Architectural View of Usability Patterns," *Proc. INTERACT Conf.,* 2003.

[16] A. Seffah, T. Mohamed, H. Habied-Mammar, and A. Abran, "Reconciling Usability and Interactive System Architecture Using Patterns," *J. Systems and Software,* vol. 81, no. 11, pp. 1845-1852, 2008.

[17] N. Juristo, A.M. Moreno, and M. Sánchez-Segura, "Guidelines for Eliciting Usability Functionalities," *IEEE Trans. Software Eng.,* vol. 33, no. 11, pp. 744-758, Nov. 2007.

[18] J. Vanderdonckt, "Development Milestones Towards a Tool for Working with Guidelines," *Interacting with Computers,* vol. 12, no. 2, pp. 81-118, 1999.

[19] S. Alpert et al., HCI Patterns, http://www.hcipatterns.org, 2012.

[20] L. Carvajal, "Usability-Oriented Software Development Process," PhD thesis, Computer Faculty, Universidad Politécnica de Madrid, http://oa.upm.es/10599/, 2012.

[21] J.A. Cruz-Lemus et al., "Assessing the Influence of Stereotypes on the Comprehension of UML Sequence Diagrams: A Family of Experiments," *Information and Software Technology,* vol. 53, no. 12, pp. 1391-1403, 2011.

[22] F. Ricca, M. Di Penta, M. Torchiano, P. Tonella, and M. Ceccato, "How Developers' Experience and Ability Influence Web Application Comprehension Tasks Supported by UML Stereotypes: A Series of Four Experiments," *IEEE Trans. Software Eng.,* vol. 36, no. 1, pp. 96-118, Jan./Feb. 2010.

[23] A. Davis, O. Dieste, A. Hickey, N. Juristo, and A. Moreno, "Effectiveness of Requirements Elicitation Techniques: Empirical Results Derived from a Systematic Review," *Proc. 14th IEEE Int'l Conf. Requirements Eng.,* 2006.

[24] J. Patrick Meyer and M.A. Seaman, "Expanded Kruskal-Wallis Tables: Three Groups," http://faculty.virginia.edu/kruskal-wallis/table/KW-expanded-tables-3groups.pdf, Mar. 2008.

Laura Carvajal received the PhD degree from the Universidad Politécnica de Madrid. Her primary research interest includes the integration of usability into software development.

Ana M. Moreno is a full professor in the School of Computing at the Universidad Politécnica de Madrid. Her research interests include software engineering education, usability, and agile development.

María-Isabel Sánchez-Segura is an associate professor in the Department of Computer Science at the Universidad Carlos III de Madrid. She works on usability and intelligent organizations.

Ahmed Seffah is a professor and the director of the Human-Centered Software Engineering Group at the Concordia University in Montreal, Canada. His areas of expertise include bridge software/service systems engineering and HCI.