Introducing Usability in a Conceptual Modeling-Based Software Development Process[*]

Jose Ignacio Panach[1], Natalia Juristo[2], Óscar Pastor[3]

[1]Universitat de València
Escola Tècnica Superior d'Enginyeria, Departament d'Informàtica
Vicent Andrés Estellés, s/n 46100 Burjassot, València, Spain
joigpana@uv.es
[2]Universidad Politécnica de Madrid
Campus de Montegancedo, 28660 Boadilla del Monte, Madrid, Spain
natalia@fi.upm.es
[3]Universitat Politècnica de València
Centro de Investigación en Métodos de Producción de Software,
Camino de Vera s/n, 46022 Valencia, Spain
opastor@pros.upv.es

Abstract. Usability plays an important role to satisfy users' needs. There are many recommendations in the HCI literature on how to improve software usability. Our research focuses on such recommendations that affect the system architecture rather than just the interface. However, improving software usability in aspects that affect architecture increases the analyst's workload and development complexity. This paper proposes a solution based on model-driven development. We propose representing functional usability mechanisms abstractly by means of conceptual primitives. The analyst will use these primitives to incorporate functional usability features at the early stages of the development process. Following the model-driven development paradigm, these features are then automatically transformed into subsequent steps of development, a practice that is hidden from the analyst.

Keywords: Model-Driven-Development, Usability, Conceptual Model.

1 Introduction

Historically, many SE authors have considered usability as a non-functional requirement. Recently, however, some authors have identified several usability features strongly related to functionality [4]. These features do not only affect interfaces but

[*] This work has been developed with the support of, MICINN (TIN2011-23216, PROS-Req TIN2010-19130-C02-02), GVA (ORCA PROMETEO/2009/015), and co-financed with ERDF. We also acknowledge the support of the ITEA2 Call 3 UsiXML (20080026) and financed by the MITYC under the project TSI-020400-2011-20 .

also architecture, and are hard to deal with if they are not considered from the early stages of development. Incorporating usability from requirements elicitation is not for free. Generating usable software has a number of unwanted collateral impacts: increased complexity [4]; increased cost; increased maintenance difficulty [5].

To mellow these effects we propose including functional usability features in a model-driven development (MDD) software process. If usability is considered from the early stages of development, it can be included in an MDD method and benefit from the advantages of the MDD paradigm [10]. MDD claims that developers should focus their efforts on building a conceptual model, then the system is implemented by means of transformation rules that can be automated [7]. If we study other works in the literature, we find very few proposals to deal with usability features in an MDD method. Moreover, when it is discussed, very few precise details are given, which makes it difficult to understand how these approaches could work correctly in practical settings. Examples of these works have been developed by Tao [11] and Raneburger [9]. From our point of view, usability is as important as functional requirements, and therefore MDD methods should provide a mechanism to abstractly represent usability. In the following, we explain how we propose including usability features with functional involvement in an MDD method

The paper is structured as follows. Second section describes our proposal for adding usability features to an MDD method. Third section discusses an experiment to evaluate user satisfaction improvement applying our proposal. Finally, Fourth section presents some conclusions.

2 Incorporating Usability Functionalities in a Model-Driven Development Method

Human-Computer Interaction literature provides many different recommendations to improve the usability of a software system. In [4], authors present three groups of recommendations: (1) Usability recommendations with impact on the user interface; (2) Usability recommendations with impact on the development process; (3) Usability recommendations with impact on the architectural design. These last recommendations involve building certain functionalities into the software to improve user-system interaction. This set of usability recommendations is referred to as functional usability features. Examples of these features are providing undo and feedback facilities. A big amount of rework is needed to include these features in a software system unless they are considered from the first stages of the software development process [1]. Moreover, their inclusion from the first steps of the development process increases the complexity of the software development. To minimize the problems of including usability features with impact on the architecture, we aim to incorporate them in an MDD method. This way, we benefit from the advantages of the MDD paradigm [10]. Our approach is divided into four steps:

1. Identify the possible use ways of each usability functionality.
2. Identify the properties that configure each use way.
3. Define conceptual primitives to abstractly represent the use way properties.
4. Describe the changes that must be made to the model compiler to generate code.

First and second steps are based on interaction patterns and usability guidelines that define how to deal with functional usability features. From all the existing works in the literature, we have chosen a list of usability recommendations called ***Functional Usability Features (FUFs)*** [4] as illustrative example to apply our proposal. Each FUF was defined with a main objective that can be specialized into more detailed goals we named mechanisms. The list of FUFs and their mechanisms are shown in [4, 6]. We focus our example on the usability mechanism called *Structured Text Entry*, which belongs to *User Input Error Prevention* FUF. We select this mechanism because its goal --*help the user when the system only accepts inputs in a specific format*-- is simple enough to allow its presentation in a couple of pages. As MDD method to include Structured Text Entry, we have selected OO-Method [8]. The OO-Method has been successfully implemented in industry (INTEGRANOVA [1]). The analyst does not have to implement any code because all the code is automatically generated from a conceptual model by means of a model compiler. Next, we explain the steps of our proposal to include Structured Text Entry in OO-Method.

2.1 Identification of Use Ways

Each functional usability feature can achieve its goal in different means. We have called each such mean Use Way (UW). Each UW has a specific target to achieve as part of the overall goal of the usability feature. We propose deriving UWs from existing works in the literature (interaction patterns and usability guidelines), such as FUFs. In FUFs list, each mechanism is defined with a set of questions to identify usability requirements. Regarding our proposal, these questions can be used to identify UWs. If we focus on the Structured Text Entry mechanism, we identify three UWs:

- **Specify the input widget visualization type (UW1):** This UW aims to specify the format of the input widget to help the user and it is derived from the usability mechanism question, *Which is the format of input arguments?*
- **Mask definition (UW2):** This UW aims to stop the user from entering data that is not in a valid format and it is derived from the usability mechanism question, *What guidance should the user receive to enter the input in the required format?*
- **Default values (UW3):** This UW aims to provide the user with guidance on which format to use to enter data and it is derived from the same question as UW2.

2.2 Identification of Properties

We have called the different UW configuration options to satisfy usability functionalities as **Properties**. In this second step, we identify properties also from the questions used in the usability mechanism definition. Focusing on the UWs derived from *Structured Text Entry*, we have identified the following properties:

- UW1 has only one property: **Type of input widget (UW1_P1)**. This property aims to define how the user will visualize input arguments and it has been derived from the mechanism question, *which is the format of input arguments?*
- UW2 has two properties: **Widget selection (UW2_P1):** This property aims to specify the widgets that need a mask and it has been derived from the mechanism

question, *which widgets require a specific format for their data*?. **Regular expression (UW2_P2):** This property aims to define the regular expression that specifies the mask and it has been derived from the mechanism question, *which is the required format for the widget?*

- UW3 has two properties: **Widget selection (UW3_P1):** This property aims to specify the widgets that need a default value and it has been derived from the mechanism question, *which widgets require a default value?* **Definition of the default value (UW3_P2):** This property aims to define the default value and it has been derived from the mechanism question, *which is the required default value?*

2.3 Definition of Conceptual Primitives

The conceptual model of the MDD method needs to be enriched to support the UWs. This step involves verifying whether or not there are already conceptual primitives in the MDD method that represent a property. If there is no conceptual primitive already to represent a property, the conceptual model needs to be expanded to ensure the required expressiveness. In order to include *UW1_P1* property in OO-Method, we need to enrich the OO-Method conceptual model with new primitives that represent the different widget types. For example, a numbered list can be represented with a ComboBox or with a RadioButton. *UW2* and *UW3* are already supported by the OO-Method conceptual model and they do not involve new conceptual primitives.

2.4 Changes in the Model Compiler

The changes in the model compiler aim to implement the code derived from new conceptual primitives. These changes involve adding new attributes, services and classes in the generated code. In our example, the only change to be made to the model compiler is to include *Specify the input widgets visualization type (UW1)*. This change has the aim of generating the code that implements the type of widget specified by means of conceptual primitives.

3 A Lab Evaluation

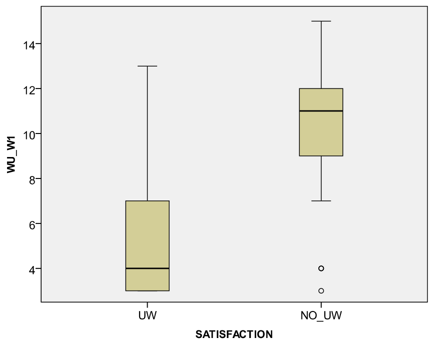
We have carried out an empirical evaluation with 66 subjects using a Web application for car rental. The users of this system are the employees of offices all over the world. This Web application has been fully developed using INTEGRANOVA [1]. The UWs not supported by INTEGRANOVA were manually included in the generated code. We included a total of 7 UWs. The aim of this evaluation is to study whether or not the end-user perceives the benefits of including UWs in the system. If the answer is positive, the effort to include UWs in an MDD method is justified, since UWs will improve the end-users' approval of the software. We divided the experimental subjects into two groups: subjects that interact with the system without UWs and subjects that interact with the system including several UWs.

We identified the following null hypotheses:

- $H1_0$: The satisfaction for the users that interact with UWs is the same as the satisfaction for users that interact without UWs.
- $H2_0$: The time for the users that interact with UWs is the same as the time for users that interact without UWs.

There are two response variables [3] in the experiment. One variable is called *user satisfaction level*. This variable measures whether or not the user is satisfied with the interaction and it is measured by means of a five-point Likert-scale questionnaire. The other variable is *time to finish* the tasks. This variable measures how long it takes the user to perform the experimental tasks. This is measured timing the seconds needed to finish the tasks. There are two factors [3] in the experiment. One factor is *use of UWs*. This factor involves studying the Web application with UWs and without them. The other factor is *previous experience of applications generated with INTEGRANOVA*. We combined both factors across the subjects to see how they affect the response variables.

We had 22 subjects with experience in INTEGRANOVA; 11 interacted with UWs and other 11 without UWs. We had 44 subjects without experience in INTEGRANOVA; 22 interacted with UWs and other 22 interacted without UWs. We analyzed the data using two methods: ANOVA and box and whisker plots. First, the ANOVA results show that the satisfaction of subjects strongly depends on using UWs (p-value=0,001 for most studied UWs). Moreover, the ANOVA analysis shows that there is no relationship between: (1) *User satisfaction level* and *previous experience of applications generated with INTEGRANOVA* (p-value=0.558 for most studied UWs)*; (2) Time to finish the tasks* and *use of UWs* (p-value=0.628); (3) *Time to finish the tasks* and *previous experience of applications generated with INTEGRANOVA* (p-value=0.057). Second, the box and whiskers plots illustrate the median and quartile for both response variables (*user satisfaction level* and *time to finish tasks*). Figure 1a shows the plot that compares *user satisfaction level* when using and not using the Use Way called *Warning message (UW_W1)*. Figure 1b shows the box and whisker plot for *time to finish the tasks* with reference to *the use of UWs factor*.



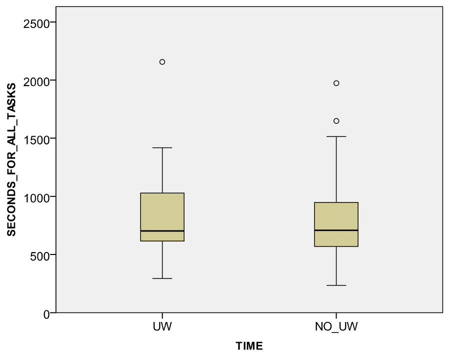


Fig. 1. a) Box and whisker plot for *user satisfaction level* with and without UW_W1 **b)** Box and whisker plot for *time to finish the tasks* with and without UWs

According to our analysis, we state that UWs generally improve user satisfaction. Moreover, satisfaction does not depend on experience in the use of INTEGRANOVA applications. So, we reject the hypothesis $H1_0$. With regard to the time hypothesis ($H2_0$), the analysis shows that time is independent of interacting with or without UWs. Moreover, there is no difference between the time of the experts in INTEGRANOVA applications and beginners.

4 Conclusions

In this paper, our aim is to incorporate functional usability features into an MDD process, benefiting from the MDD advantages and minimizing manual implementations. The method we propose can be easily applied to other MDD methods than OO-Method. Use ways and properties can be directly applied to other methods; while the conceptual primitives and the changes in the model compiler depend on a specific MDD method, since conceptual model and model compiler are exclusive of a MDD method. The difficulty of applying our proposal to other MDD methods depends on the expressiveness of their conceptual models. OO-Method has an Interaction Model, which facilitates the inclusion of new conceptual primitives to represent interaction features. However, MDD methods with less expressiveness to deal with interaction would require adding more conceptual primitives to represent UWs.

5 References

1. Bass, L. and John, B. Linking usability to software architecture patterns through general scenarios. The journal of systems and software, vol. 66. pp 187-197, (2003).
2. CARE Technologies S.A. http://www.care-t.com
3. Juristo, N. and Moreno, A., Basics of Software Engineering Experimentation: Springer, (2001).
4. Juristo, N., Moreno, A., and Sánchez, M.I., "Analysing the impact of usability on software design," in Journal of Systems and Software, vol. 80, pp. 1506-1516, (2007).
5. Lawrence, B., Wiegers, K., and Ebert,C. "The top risk of requirements engineering," in IEEE Software, vol. 18, pp. 62-63, (2001).
6. List of FUFs: http://hci.dsic.upv.es/FUF/FUFList.html
7. Mellor, S.J., Clark, A.N. and Futagami, T."Guest Editors' Introduction: Model-Driven Development," in IEEE Software, vol. 20, pp. 14-18, (2003).
8. Pastor, O., Gómez, J., Insfrán, E. and Pelechano, V. The OO-method approach for information systems modeling: from object-oriented conceptual modeling to automated programming. Information Systems, vol. 26,507-534, (2001).
9. Raneburger, D., Popp, R., Kavaldjian, S., Kaindl, H., and Falb, J. "Optimized GUI Generation for Small Screens," in Model-Driven Development of Advanced User Interfaces, vol. 340/2011: Springer, pp. 107-122, (2011).
10. Sendall, S., and Kozaczynski, W., "Model Transformation: The Heart and Soul of Model-Driven Software Development," IEEE Software, vol. 20, pp. 42-45, 2003.
11. Tao, Y., "An Adaptive Approach to Obtaining Usability Information for Early Usability Evaluation," Proc. of IMECS, (2007).