Use of Abstraction during User Interface Development

A Position paper

Ebba Þóra Hvannberg
University of Iceland
Hjardarhaga 2-6, 107 Reykjavik, Iceland
+354 525 4702
ebba@hi.is

ABSTRACT

This paper postulates a thesis claiming that abstraction is an essential part of communication during user interface development, that models are a way of expressing those abstractions and that user interface developers and software engineers need the same language for communication. Motivated by described myths and desired model characteristics stated in the literature, several counterarguments and arguments are given to the thesis, backed up with results from empirical research studies. The paper concludes with a plan of action to bring the thesis forward.

Categories and Subject Descriptors

H5.2 [**User Interfaces**]: *models*

General Terms

Human Factors

Keywords

Models, Abstraction, Development

1. INTRODUCTION

During software development, good communication within a development team and between a team and the stakeholders is essential. Many development lifecycle models have been suggested, and since participatory design, most if not all lifecycle models have emphasized inclusion of users. Recent agile models include two characteristics which involve users; writing some kind of user stories and letting the buyer of the product decide upon the next features in the product. Agile methods also stress that communication within teams are important, but they do discourage heavy documentation, processes or tools usage. Communication within a team is sometimes between different roles. The gap between software engineering and user interface development has been addressed to an extent in the literature and the conclusion is that whatever method is used the difficulties in communication between the software developers and usability specialists must be tackled [1]. We can all agree that communication is important, but how, what and why? Engineers have long communicated by means of mathematics, structural (architectures) and behavioural models (electrical engineers). They communicate about materials, structures of buildings, input and output of processes or systems. Computer scientists on the other hand express things with logic or computer programs. Because it seems so easy to change programs or write new ones, unlike concrete materials such as metal or cement, programmers think that modeling is not necessary, and in the race for fast products to market, they skip the preparation and planning and dive right into the implementation [2].

Because of inherent complexity of software, or maintenance, computer scientists tend to abstract from details for easier comprehension during development. Much of the effort of research in software engineering has been on how to communicate and articulate this abstraction. Early, this abstraction appeared as functions, with input and output as descriptions of change of states of the machine, then as user defined data structures. This was followed by entity-relationship models which had a strong influence on data modelling. Finally, since few decades, abstraction has been dominant in object-orientation, where abstraction occurs in forms of inheritance and encapsulation. Reuse was the anticipated return on investment of abstraction, initially with the concept of classes but when that did not meet expectations, recent developments have centered more on components and services as a form of abstraction.

There have been different suggestions of specializing descriptions of user interfaces from descriptions of software in general, such as patterns for user interfaces, frameworks and classes to user interface programming [3]. Instead of specialization from general software models, others have designed models for user interfaces independent of software development, such as cognitive models [4] . With the advent of the web as a platform, specific languages have been developed and engineering tools developed such as Web-ML and OO-H [5]. There have thus been countless attempts to devise specific models for user interaction, and while they are useful as such they will probably not be widely used by software developers, or in an interdisciplinary team of software developers and user interface designers [1]. Those models which are based on software engineering models such as UML-Web based Engineering (UWE) are perhaps more likely to be used in such teams [6].

From time to time, or should I say continuously, the research community has been trying to discover why developers refrain from using models as abstractions of software, be it informal or formal. As in the case of formal methods, scientists have tried to elicit myths and refute them [7]. Sometimes these myths have been denied with facts, sometimes with general arguments. Myths are drawn from speculations, here say or common knowledge in the area. While it is useful to gather such tacit knowledge present it explicitly, we need to conduct more empirical research investigating usage of models. A few such studies and my own results have motivated several statements of arguments for user interface modeling. The next section states a thesis, followed by counter arguments and arguments for that thesis. The paper concludes with a plan of actions.

2. THE THESIS

I postulate that a good way to communicate user interface designs, and hence results of usability evaluations, is through abstractions. I postulate that models as a form of abstraction are the best way to discuss and argue about a user interface development. The models are created during the requirements, design activities, used to assist during evaluation, used to express, understand and even predict results of an evaluation, and used to redesign a user interface to fix a usability problem uncovered during evaluation. Finally, I claim that for the task we need **software development models** in order to bridge the gap between user interface and software designers.

3. COUNTER ARGUMENT

Probably, not everyone agrees with the stated position. In the following, let us examine some of the counter arguments, some of which are motivated by the characteristics of good models, i.e. *abstraction, understandability, accuracy, predictability and inexpensiveness* [2], others which are motivated by myths stated in the literature [1, 7].

3.1 Working software over comprehensive documentation

One of the principles of agile development is working software over comprehensive documentation. Daily face to face meetings and frequent changing of roles or activities is meant to make up for lack of documentation. Knowledge is tacit and not explicit [8].

Modelling is a process which aids in describing and abstraction what is to be built. In support of this counterargument Ambler [9] refers to Constantine [10] who says that it is a misconception that agilists do not model. The truth is, Ambler states, that they do model, but that they discourage extensive modelling up-front but encourage active modelling along the way. Ambler supports this by referring to agile methods' literature, but also acknowledges that the models are sometimes rather informal.

Further, some of the misunderstanding of models is that their impact is to be mainly achieved through the end product. Instead, modelling is a dynamic activity and much is gained by the interaction which the modelling activity facilitates

3.2 Models are difficult to create and few know how to make them

Not many modeling languages have been created exclusively for user interface design or for that matter software development. The predominant one for software development is UML and it is quite large containing a number of different model types. The types of problems architects describe are scattered information among different model views, incompleteness of models, disproportion, i.e. more details in some parts than others and inconsistencies between teams. Furthermore, architects claim that models are sometimes used informally and there are a lack of modeling conventions [11]. A study on the use of UML demonstrated that those with more UML experience used it more extensively than those with less experience, suggesting that analysts need time to learn how to use the UML language well [12].

While I agree that modeling can be an intricate activity, I don't think it is the models themselves that are difficult to create, but it is the activity of abstraction which is hard. Successful user interface designers will always need to learn how to abstract. Some will learn it through modeling; others will learn it implicitly as they gain experience. With models they are forced to do it but they can avoid it they don't use models, with unpredictable results.

3.3 Creating models are costly and not worth the effort

Creating models, especially if supporting tools are unavailable, can be a difficult and time consuming effort. Not only are models difficult to create but also evolve ensuring that the models are synchronized with the implementation. A survey says that 52.5 percent of practitioners finish modeling when the model is complete, 33.8 percent of practitioners say that a model is done when it has passed a review or an inspection, and 32.8 percent of practitioners say that the deadline is the stopping criterion. Whereas the completeness of a model is more often a stopping criterion in larger projects, a deadline is more often a halting criterion for smaller projects [11]. These numbers tell us that models are created in different ways, and in the cases where the models are not complete, developers do not take full advantage of the benefits of models, namely model driven development where code is automatically generated from models [2].

A study we conducted recently showed that over 30% of the defects could be blamed on faulty dialogue or navigational design, yet only a few of those defects were fixed [13]. Why? We speculate that the reason may be that it was estimated too difficult to fix the usability problems because the solutions required a revised user interface architecture and hence were too costly or even too difficult to make.

Our conclusion, from our own and other research studies, is that it is very costly not to create models, and that unless models are complete, their full benefits are not reaped.

3.4 Models are limited to describing those characteristics of user interfaces which do not concern presentation

Models, especially very abstract ones, do not capture experience very well. To understand emotional experience, we need a detailed contextual implementation.

A survey among 171 analysts showed that of seven different types of UML diagrams and narratives, class diagrams were used most frequently, with 73% of the analysts saying that they were used in at least two-thirds of their projects. Use case diagrams were second, use case narratives fourth (44%), but statechart diagrams came sixth, with less than 30% of the analysts saying that statecharts are used in at least 2/3 of the projects. On the other hand when analysts were asked to mark those diagrams which were never used, class diagrams ranked the lowest with only 3% to 25% for collaboration diagrams, ranked the highest [11].

In this same survey, respondents were asked for the usefulness of the different diagrams. Interestingly, whereas statechart diagrams were used much less frequently than class diagrams, they ranked second in usefulness after class diagrams.

If we were to ask user interface developers, I speculate that class diagrams are only useful for conceptual modelling, but activity diagrams and then state charts diagrams would be ranked higher in terms of providing new information not found in use case narratives.

Conceptual modelling is still very useful in user interface design. Our study showed that around 23% of defects uncovered could be attributed to wrong conceptual models [13]. As we see in UML there are a number of different types of diagrams and this is what we should aim for in user interface modelling, but we need to link the different models together such as the presentation models to the structural and behavioural models, or else the developers will complain that there is a disconnect between the models.

3.5 Users do not understand models

In a user-centered development, it is imperative to involve users at all stages of development. It is also critical to include a multi-disciplinary group of experts. Therefore, the communication language needs to be familiar to all. Undeniably, artifacts such as low-fidelity prototypes, story boards, scenarios and use case narratives are very accessible to users, and countless research papers have claimed the usefulness of informal models of user interaction design such as scenarios and prototypes.

The results of a study on how UML is used, partly refutes this counterclaim. While the study' results reveal that stakeholders are most likely to use use case narratives and use case diagrams, clients are involved in developing, reviewing and approving other components more than expected. All of the clients interviewed in the study welcomed the use of UML and some even showed insight into its usage [12]. As expected, client involvement is highest with use case narratives, 76%, and lowest for statechart diagrams.

What is worrying is that models which are not useful with clients may be useful for programmers, thus creating a gap between the two groups.

4. ARGUMENT

In this section we restate our claims and support them.

4.1 Abstraction is key to communication

With abstraction we are able to discuss main interactions and principles in the software without burying it in too many details. Abstraction makes it easier to plan, verify and design. Abstraction allows us to present different views of the user interaction.

4.2 Models are a good way to communicate during user interface development

Sketches, scenarios or storyboards are all different types of models, since they describe the real end product but leave out some of its details. Diaper states that "HCI is an engineering discipline and therefore must model the real world that is assumed to exist, notwithstanding how poor and partial might be our models of it." [14]. Diaper emphasises the importance of task models since they describe a series of events or activities in time. He doesn't exclude other models but says that they play a lesser role. Seffah and Metzker acknowledge that task models are widely used in the user interface community but warn that they may describe functionality more than usability, thus not fulfilling the objectives of the user interface developer.

One of the desirable characteristics of models is that they should be predictive. Prediction does not only include foreseeing the behaviour of the user and the system through simulation, but also modelling of the development activity itself and not just the artefacts. With increased emphasis on approaches for the whole lifecycle, including maintenance, we need to include models for evaluations of user interfaces. Modelling evaluation results should help us predict whether a defect is likely to be fixed, whether an evaluator is likely to uncover defects, whether components are likely to be faulty etc.

4.3 Software development models can serve user interaction design and other components' designs

In communication between people a disagreement is often due to misunderstanding. We say that people don't speak the same language. To close the gap between software engineers and user interaction designers they need to speak the same language. Different dialects can be permissible but not different languages.

5. CONCLUSION

Current research gives evidence that user interface designers need better help in their work. The number of defects found and the increasing criticality of user interfaces demands that we continue searching for better ways to communicate and apply abstractions in interaction designs.

The counter arguments stated in this position paper are however real threats to this believe. I think these threats can be lessened with the following plan of action:

1. Develop a domain specific modelling language for user interface design which can be used by an interdisciplinary team of user interface designers and software developers.

2. Offer tutorials and develop body of knowledge for user interface modelling as an abstraction and communication activity.

6. REFERENCES

1. Seffah, A. and E. Metzker, *The obstacles and myths of usability and software engineering.* Commun. ACM, 2004. **47**(12): p. 71-76.
2. Selic, B., *The pragmatics of model-driven development.* Software, IEEE, 2003. **20**(5): p. 19-25.
3. Myers, B.A. and M.B. Rosson, *Survey on user interface programming*, in *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1992, ACM: Monterey, California, United States.
4. de Haan, G., G.C. van der Veer, and J.C. van Vliet, *Formal modelling techniques in human-computer interaction.* Acta Psychologica, 1991. **78**(1-3): p. 27-67.
5. Abrahão, S., et al. *A Model-Driven Measurement Procedure for Sizing Web Applications: Design, Automation and Validation* in *MoDELS* 2007: Springer-Verlag
6. Koch, N. and A. Kraus. *The expressive power of UML-based engineering.* . in *Second International Workshop on Web Oriented Software Techonlogy (CYTED)*. 2002.
7. Bowen, J.P. and M.G. Hinchey, *Seven more myths of formal methods.* Software, IEEE, 1995. **12**(4): p. 34-41.
8. Pikkarainen, M., et al., *The impact of agile practices on communication in software development.* Empirical Software Engineering, 2008. **13**(3): p. 303-337.
9. Ambler, S., *Tailoring Usability into Agile Software Development Projects*, in *Maturing Usability, quality in Software, Interaction and Value*, Effie Lai-Chong Law, Ebba Thora Hvannberg, and G. Cockton, Editors. 2008, Springer-verlag London. p. 75-95.
10. Constantine, L. *Process Agility and Software Usability: Toward Lightweight Usage-Centered Design. Accessed on April 25, 2006.* 2001 [cited 2009; Available from: www.foruse.com/articles/agiledesign.pdf
11. Lange, C.F.J., M.R.V. Chaudron, and J. Muskens, *In practice: UML software architecture and design description*, in *Software, IEEE*. 2006. p. 40-46.
12. Dobing, B. and J. Parsons, *How UML is used.* Commun. ACM, 2006. **49**(5): p. 109-113.
13. Law, E.L.-C., et al., *Impacts of Classification of Usability Problems (CUP) on System Redesign* in *Usability and User-Centered Design in Software Development: Case Studies and Real Life Applications*, Ann Blandford, et al., Editors. 2010, in review, IGI
14. Diaper, D., *The discipline of HCI.* Interacting with Computers, 1989. **1**(1): p. 3-5.