Innovative User Interface Engineering

Pradip Peter Dey¹, Bhaskar Raj Sinha², Gordon W. Romney³, Mohammad Amin⁴ and Hassan Badkoobehi⁵

Abstract—User interface engineering, critical to the development of stellar applications, is one of the most challenging areas given the diversity of knowledge, ideas, skills and innovations needed for building smart interfaces in order to succeed in today’s rapidly paced and tough, competitive marketplace. Many engineering aspects including graphical, analytical, intuitive, artistic, mathematical, psychological and programming models are seriously considered in the development process of a user interface. This paper critically examines some of the past practices and suggests a set of principles for developing alluring user interfaces. It also shows how UML use case diagrams can be enhanced. The improved design constructs of an augmented UML view are presented with examples for highlighting and clarifying important engineering issues.

Keywords—Design principles, software modeling, Unified Modeling Language (UML), usability.

I. INTRODUCTION

The challenge of building smart user interfaces is often considered to be “beyond the reach” of ordinary software developers, particularly, when compared to the repeated achievements of an extraordinary genius such as Steve Jobs of Apple, Inc. However, neither Steve Jobs, nor anybody else, has described the innovative process by which the challenges can be overcome and smart user interfaces created. This paper critically examines a number of the past practices and suggests a set of principles upon which future innovative user interface engineering can be guided. Every time a human uses a digital product, machine or tool the interaction takes place through a machine-to-human boundary or interface. If the interface is correctly structured, then the user is likely to have a satisfactory experience which invites the user back again, and again. Designing elegant user interfaces for complex computational systems presents formidable challenges [1]-[6]. The employment of use case analysis in the software development process has been increasingly utilized because use cases reduce complex systems to manageable aspects [7]. Usability questions in design are drawing more attention than any others in recent years [6]. Software design, including user interface design is based on current best practices since practicing engineers have developed useful strategies based on past experience [1]-[12]. Support for context-aware user interfaces is evolving to a level where it becomes feasible even in large systems [13]. User interface quality is difficult to assess, and, yet, an emergent discipline is attempting to do so [1], [2], [6]. A good user interface is truly appreciated only when it is integrated with a smart total system architecture, including hardware and software, that renders a useful service. User interfaces cannot be considered in isolation from the entire integrated system. Software development has often been considered as one of the most challenging processes of modern technology. Some approach it from a scientific perspective while others treat it in an artistically creative manner. Over the decades, a multitude of approaches to software development have been proposed. These approaches are often presented with effective metaphors. Donald Knuth initially suggested that software writing is an art [14]. David Gries argued it to be a science [15]. Watts Humphrey [16] viewed it as a process. In recent years, practitioners have come to realize that software is engineered [1]-[2], [17]-[22]. Engineering techniques have steadily improved the product quality in software development including the quality of user interfaces [1], [2], [5], [6].

The significance and role of user interface engineering in product design has recently been in focus in many of the highly successful interactive systems [6]. Certain aspects of user interfaces including graphical aspects could not be adequately developed before object oriented programming. Indeed, it has become easier to design and implement a Graphical User Interface (GUI) with object oriented concepts and languages. The Unified Modeling Language (UML) has made significant contributions in representing software design including certain aspects of usability [8]. The UML includes modeling of use case aspects in various views including the use case view [8]. However, the UML does not include modeling and representation of GUI. This paper critically examines important development issues and the UML use case view [23].

In an iterative process, practitioners have come to realize that a complex system with smart GUI elements cannot be built in one pass. In an iterative process, after requirements analysis, an initial software design is constructed. Next, the requirements analysis is revised on the basis of a combination of software design reviews, new or
changed requirements, or other factors, which in turn lead to the next software revision design. That is, the iterative process of development, or the spiral process model [22], is found to be one of the most productive software development processes. Certain aspects of software are such that after an initial assessment, iterative refinements lead to significant improvements. One of the major benefits of the iterative process is the improvements made in the design of user interfaces through successive iterations [24]. The current study is based on the iterative scheme shown in Figure 1, where software design and modeling is followed by design review and evaluation.

![Figure 1: Iterative Design and Review](image)

Figure 1 shows an iterative process of design and review in the central core with solid double arrows which allows developers to start with a highly abstract conceptual design after an initial requirements analysis. The details can be gradually added in successive iterations. The dotted arrows show other viable alternatives. User interface development requires adjustments and refinements that are best done in iterations [1], [2], [16], [17], [22], [24]. Often defects are found during the review or evaluation process and such defects need to be corrected. The design may start with just a few elements with some possible defects; other elements may be incrementally added, and new defects identified may be corrected successively. The design review may be performed by the designer or by external reviewers formally or informally.

III. USE CASE VIEW

“Separation of concerns” is a fundamental premise of Software Engineering proposed by Dijkstra [31] and arguably leads to multiple views of a software product. Separation of concerns is useful to software engineers as long as interactions among system elements are regular and predictable. The authors posit that the segmentation of the whole system into multiple views motivated by separation of concern should provide an undistorted total picture of the integrated system when the views are put together. However, care must be exercised because multiple views may over simplify the system without accounting for interactions of the system elements. The rules of composition need to be spelled out consistently because the whole picture needs to become clear when the multiple views are composed together. According to UML2.0, there are nine views for describing different aspects of software [8]. The views are: use case view, static view, design view, state machine view, activity view, interaction view, deployment view, model management view, and profile. A view is a subset of the UML modeling constructs representing certain aspects of the software [8]. Each view is illustrated in [8] with one or more diagrams that present the main features of the view. The UML use case view is presented with a use case diagram for capturing use case features. The use case view is well-utilized due to the role use cases play in defining requirements analysis and management as well as user interface design [7]. Use cases clarify many important software issues early in the development process so that some progress in designing the software can begin [1], [7] with an engineering process. The basic issue with the use case view is that it is too narrowly defined in the UML. “The use case view models the functionality of a subject (such as a system) as perceived by outside agents, called actors, that interact with the subject from a particular view point” [8: page 34]. The perception of the outside agents such as end users is primarily mediated through an interface such as a GUI. However, UML use case view does not explicitly deal with user interfaces or interfaces between the actors and the use cases. The diagram that characterizes the use case view is the use case diagram which presents the major use cases in a box with the actors outside the box to indicate that the actors are external users of the current software. One of the problems with the use case diagram is that it leaves out interfaces with the actors although each actor is shown to be using one or more use cases. For the purpose of illustration, consider a sample use case below, in Figure 2.

The following initial requirements description characterizes the start of a small software project: Develop a software system for computing the volume of two types of storage units: box-storage and cylinder-storage. Users should be able to enter inputs interactively using a Graphical User Interface (GUI). After performing the initial requirements analysis, software engineers would discover that the system must be web-based and should be available 24/7. Users should be able to access the software without any login ID. The system should be easy to maintain using web-based tools. The functional and non-functional requirements would be properly analyzed. Finally, a software requirements specification (SRS) document would be prepared and use case driven [7]. The use
The UML use case diagrams properly show use cases with ovals within the system boundary represented by a rectangle. One of the issues with a UML use case diagram, such as the one shown in Figure 2, is that it ignores the interfaces between the actors and the use cases although it depicts the actors as stick figures outside the current system boundary. For example, Rumbaugh, Jacobson and Booch [8: page 34] present a use case diagram for a subject called box office with four actors without any interfaces. In order to model functionality of the system as perceived by the actors, interfaces appropriate for the given actors need to be incorporated. This research proposes that use case diagrams include the appropriate interfaces. Thus, the use case diagram given in Figure 3 is recommended for the sample software project mentioned above. Note that the interfaces are shown with dotted rounded rectangles. Such interfaces are referred to as the general interfaces in order to distinguish them from specialized interfaces in UML 2.0 such as provided interfaces and required interfaces [8]. If an interface is to be developed as a part of the current software system, then the interface is shown within the system boundary; otherwise, it is shown outside the system boundary. In order to refer to the interfaces, they are numbered. If an interface is a graphical user interface (GUI), then it is marked with “GUI”. In addition, when one general interface includes another, it may be marked appropriately. If there is a third general interface that includes the first, then “3 ⊆ 1” can be shown in the third interface. Having general interfaces in the use case diagram intuitively and logically supports the idea that user’s perception about the functionality is modeled appropriately in the use case view. When the actor is a human user, the general interface may be a GUI. It is the role of GUIs that is not adequately detailed in the UML modeling techniques leading to a high degree of confusion for the development of modern interactive systems.

In addition to use case diagrams, the augmented use case view should have general interface diagrams. Without such a diagram concerns about user interfaces are grossly ignored and interactions among system elements are not appropriately accounted for. Without interface diagrams, the standard UML [8] misses information vital to the success of a modern software system. It also misses to give a comprehensive account of the software which is expected to be a composition of the standard UML views. We are flexible about the notations of the general interface diagrams. Two main alternative notations for the general interface diagram are (1) screen shots from a prototype, and (2) abstract graphical representation of major interface elements. We show the former notation in the general interface diagram given in Figure 4 for the general interface 1 of Figure 3.
Jason Hong [6] asks an important question: “how do we effectively incorporate great design into products?” Currently, we cannot incorporate GUI design into standard UML based documentation. The role of the UML in modeling can be enhanced by appropriately accounting for the perceived functionality of a system by providing the augmented use case view along with general interface diagrams. This is true because the augmented use case view includes general interfaces in its use case diagram between the actors and the use cases. The functionality is perceived by the actors as it passes through the general interfaces. The balance between abstraction and details can be appropriately achieved in the general interface diagram as the interface elements can be added incrementally. “Software engineers and programmers are often competent users of the technology . . . All too often, however, they do not use this technology in an appropriate way and create user interfaces that are inelegant, inappropriate and hard to use” [2]. The augmented use case view puts extra emphasis on modeling user interfaces. This promotes focusing on many other aspects of user interfaces such as maintaining input mechanisms the same throughout the application.

It may be argued that interfaces are adequately treated in the UML design view and that augmentation of the use case view is not required. This argument is found to be inadequate, because the design view simply places the provided and required interfaces with their appropriate components. Extra emphasis is needed for showing the details of interfaces of certain types such as GUIs. Modeling GUIs for interactive systems has become increasingly important in the past two decades [1], [2], [6], [25]. Separation of concerns [26] motivates modular design where a software system is decomposed into components; however, well-defined interfaces need to be specified among the components. GUIs may be required for human interactions with the components. The main confusion with the UML is that its presentation of software aspects totally disregards GUIs. A visual modeling language such as the UML cannot achieve an important goal of designing and creating a GUI with appropriate attention to GUI design. In addition, software engineering education with the UML requires guidance for learners so that different views together would be able to define the complete software system compositionally. Due to missing elements such as GUIs, the UML provides a fragmentary view of the software which is inadequate for any account of the integrated whole system. The proposed augmented use case view is designed to fill the gap. Reasoning with the augmented use case view is better than with traditional use case view, because the functionality of the system, as perceived by the actors, is more reasonable by including the general interfaces. Engineering practices and design activities with the general interface constructs may also encourage and promote learning about user interfaces which is valuable for students in educational settings and academic environments.

IV. DESIGN PRINCIPLES

In this section, we propose a set of design principles for developing user interfaces. Jason Hong [25] observes that “Apple tends to design by principle rather than from data.” Human Computer Interaction (HCI) data along with use case scenarios may help in understanding some aspects of user interfaces. However, these may not help much if the goal of the design is to present an innovative solution to exceed all expectations. HCI data are useful for accomplishing the more modest goal of “meeting expectations”. Advanced design principles along with effective strategies may lead to innovative user interface design. The following user interface design principles include the principles discussed by Hong [25] in the context of Apple, plus others that we found to be valuable for innovative solutions.

1. Examine promising alternatives from the widest range of possible alternatives in order to provide the best user experience through integration of various features including hardware, software, artistic, mathematical and intuitive aspects.
2. Let subject matter experts play a leading role in all phases of design.
3. Push the design-review-design cycle to its limits.
4. Consider separation of concerns in order to deal with all interactions among system elements.
5. Consider design principles as well as HCI data and user experience for innovative user interface solutions.
6. Include only those action features which are intuitively learnable; transform others to this category or to an automated category.
7. Maximize cohesion and minimize coupling among components.
8. Include error prevention and simple error handling.
9. Present user interface design at multiple levels of abstraction

For innovative user interface solutions, designers need to consider unusual alternatives in addition to the obvious ones. With reference to principle 1 suggested above, it is important to mention that quick design under time pressure leads to consideration of only a few obvious alternatives missing innovative but unapparent alternatives. Apple came up with brilliant user interface solutions that were missed by others in the same field.

Principle 2 is thoroughly discussed by Hong [25] with an example where contributions of subject matter experts are explained with an example of an experienced photographer. Experienced subject matter experts often quickly determine what will, or will not, work in a given context.

Principle 3 suggests that improvements can be achieved by repeating the design-review-design cycle for a complex system. We have suggested an iterative design-review-design cycle in Figure 1. Through an iterative process a designer may exhaustively explore many alternatives by critically examining her own designs.

Principle 4 is based on a traditional strategy for dealing with complexity [1], [26]-[32]. The complexity of a system becomes increasingly difficult if the degree of interactions among its elements become unpredictable. As the concerns
are separated, their relations become properly understood and, consequently, their interactions become predictable. Principle 5 is based on a commonsense integration of HCI factors [27], user experience, and other advanced design principles. A good study of user groups helps in the understanding of user interface aspects which may stimulate innovative user interface constructs [27], [28], [30].

Principle 6 basically suggests that users should not be burdened by difficult learning tasks. If there are tasks that are not easy to learn, the designer should try to automate them as much as possible.

Principle 7 is discussed in most textbooks [1]-[2]: it is related to Principle 4 because loosely coupled systems have advantages over tightly coupled systems. Interactions among components of a tightly coupled system are often unmanageable.

The idea of Principle 8 is based on Ben Shneiderman’s suggestion [27] that when users are prone to make errors, an automated or easy recovery process should be used to prevent the error from occurring.

Principle 9 makes sure that design is expressible in multiple levels of abstraction without significant loss of clarity. When one level of abstraction is transformed into another level, consistent interpretations should be applicable. Presenting user interfaces in multiple levels makes sure that no inconsistencies exist. In addition, the gap between high level design and low level design should be eliminated in the final phase. It is to be noted that the proposed design principles do not contradict with the various versions of the UML [8], [32], [33] or the enhancements suggested above. The proposed design principles combined with augmented use case view have great potentials for smart user interface design.

V. CONCLUSION

Innovative user interface design is not entirely based on data as observed by Hong [6]. As user interfaces become increasingly important, a set of principles that direct selective iterative design techniques are considered helpful in developing innovative user interface engineering. Consequently, it is reasonable to expect that various aspects of user interface modeling and design might be, procedurally, systematically reviewed and revised in an iterative cyclical process. The UML use case view is reviewed and suggestions are made for augmenting the use case view. Research of user experience (UX) is a critical component of use case development [30]. The enhancements suggested in this paper are most applicable in dealing with GUI aspects that are missing in the standard UML. Without GUI related constructs, the UML appears to be deficient and, therefore, the addition of general interface diagrams is suggested. This addition significantly enhances software modeling in UML. Design principles suggested here have the potential to help in the development of smart user interfaces. Some of these principles were inspired by the research from one of the most successful user interface design companies [6], [25].

ACKNOWLEDGMENT

The authors gratefully acknowledge the help and/or encouragements received from John Cicero, James Jaurez, Arun Datta, Alireeze Farahani, Debra Bowen and many others during the preparation of this paper and/or the research reported in it.

REFERENCES

Dr. Pradip Peter Dey is a Professor at National University, 3678 Aero Court Dr., San Diego, CA, 92123, USA. He is a Lead Faculty for the MS in Computer Science program, School of Engineering and Computing. His research interests are computational models, software design, mathematical reasoning, visualizations, User Interfaces and Computer Science education. Phone: 858-309-3421; email: pdey@nu.edu.

Dr. Bhaskar Raj Sinha is a Professor at National University, 3678 Aero Court Dr., San Diego, CA, 92123, USA. He is the Lead Faculty for the BS in Information Technology Management program, School of Engineering and Computing. Dr. Sinha has more than 25 years of research and teaching experience in industry and academia. His interests are in Mathematical Reasoning, Digital Systems, Computer Architecture, Technology Management, and Engineering Education. Phone: 858-309-3431; email: bsinha@nu.edu.

Dr. Gordon W. Romney, is a Professor in the Department of Computer Science, Information and Media Systems, School of Engineering and Computing, National University, San Diego, California, USA. His major research interests are Cyber Security and Information Assurance, Information Integrity and Confidentiality, 3D Graphics, Anonymization of Data, Virtual Laboratories, Secure Cloud Infrastructures, Secure Social Networks, and Agile Development using Ruby on Rails. His email address is: gromney@nu.edu.

Dr. Mohammad Amin is with National University, 3678 Aero Court Dr., San Diego, CA, 92123, USA. He is a Professor and Lead Faculty for the Master’s degree program for the MS in Wireless Communications program, School of Engineering and Computing. His major research interests are computational modeling, wireless communications, databases, sensors and engineering education. Phone: 858-309-3422; email: mamin@nu.edu.

Dr. Hassan Badkoobehi is with National University as a Professor in the School of Engineering and Computing at 3678 Aero Court Dr., San Diego, CA, 92123, USA. His major research interests are engineering education, environmental engineering, and statistical reasoning. Phone: 858-309-3437; email: hbadkoob@nu.edu.