BEST PRACTICES FOR IMPROVING USER INTERFACE DESIGN

Pradip Peter Dey, Bhaskar Raj Sinha, Mohammad Amin, Hassan Badkoobehi

Department of Engineering and ComputingNational University,
San Diego, California, USA

ABSTRACT

A rich and effective computational system must have a friendly user interface with appealing usability features that provides excellent user experience. In order to develop interactive systems with the best user experience, an innovative iterative approach to user interface engineering is required because it is one of the most challenging areas given the diversity of knowledge, ideas, skills and creativity needed for building smart interfaces in order to succeed in today's rapidly paced and tough, competitive marketplace. Many modeling aspects including analytical, intuitive, artistic, technical, graphical, mathematical, psychological and programming models need to be considered in the development process of an effective user interface. This research examines some of the past practices and recommends a set of guidelines for designing effective user interfaces. It also demonstrates how UML use case diagrams can be enhanced by relating user interface elements to use cases.

KEYWORDS

Design principles, Unified Modeling Language (UML), use case diagram, user experience.

1. INTRODUCTION

User interface design becomes iterative because after finishing an initial design the designer usually asks: Can it be made better? And the usual answer is "yes". Software design, especially user interface design, is naturally iterative, because software allows changes as long as engineers want to make changes for enhancements or experiments. After the initial design is created, software design continues to evolve through iterations, experiments with prototypes, or incremental development. We are not likely to learn much about software design from the design of physical systems such as buildings. "Because software is so malleable, software design is a continuous process that spans the entire lifecycle of a software system; this makes software design different from the design of physical systems such as buildings, ships, or bridges" [1]. User interface design is one of the most challenging areas of software engineering. The challenges of building innovative user interfaces is often considered to be "beyond the reach" of ordinary software developers, particularly, when compared to the repeated achievements of Steve Jobs and Jonathan Ive of Apple, Inc. Creating great design is not easy [2]. Great software designers have not written much about their innovative design approaches. This is one of the difficulties in understanding and replicating great design techniques [2]. There are other difficulties in learning software design, especially for complex software systems [1-3]. Software complexity is challenging since "it isn't possible to visualize the design for a large software system well enough to understand all of its implementations before building anything" [1]. An initial software design may have to be revised iteratively after the initial development phase when better insights about the complexity of the system becomes evident. The initial user interfaces of the system may play a very crucial constructive roles in the formative process. Planning for changes through iterative development may produce better results. This paper critically examines

DOI: 10.5121/ijsea.2019.10505

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 even when users intend to communicate among themselves. If the interface is correctly structured, then the users are likely to have a satisfactory experience which invites them back again and again. Designing elegant user interfaces for complex computational systems presents daunting challenges [1-10]. The employment of use case analysis in the software development process has been increasingly utilized because use cases help in reducing complex systems to manageable aspects [9-10]. Usability questions in design are drawing more attention than any others in recent years [2, 9]. Software design, including user interface design, is based on current best practices since practicing engineers have developed useful strategies based on past experiences [1-15]. Support for context-aware user interfaces is evolving to a level where it becomes feasible even in large systems [15]. User interface quality is difficult to assess, and yet, an emergent discipline is attempting to do so [1, 3, 8]. A good user interface is truly appreciated only when it is integrated with smart total system architecture including hardware and software that renders a useful service in a meaningful way. 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 given the diversity of knowledge, ideas, skills and creativity needed for building smart systems with in order to succeed in today's rapidly paced and tough, competitive marketplace. Software development can be viewed as a knowledge creation process. 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 described with impressive metaphors. Donald Knuth initially indicated that software writing is an art [16]. David Gries argued it to be a scientific endeavor [17]. Watts Humphrey [18] viewed software development primarily as a process. In recent years, practitioners have come to realize that software is engineered [4-5], [18-23]. As a result of the adoption of engineering methods, software development techniques have evolved and software product quality has steadily improved.

The significance and role of user interface engineering in product design has recently been the focus in many of the highly successful interactive systems [2]. 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 [10]. The UML includes modeling of use case aspects in various views including the use case view [10]. However, the UML does not include modeling and representation of GUI. This paper critically examines important development issues and the UML use case view and proposes an augmented use case view which is more appropriate for modern user interface modelling. It suggests that certain interface elements should be properly included in use case diagrams. It proposes some elements of modeling GUI in an intuitive language similar to UML. In addition, it presents a set of principles for developing innovative user interface features following the suggestions in recent studies [1-2].

2. DESIGN PROCESS

"Software design is an iterative process . . ." [4, page 228]. User interface based interactive software systems are designed and developed using evolutionary, iterative, agile and prototype based processes [4-5], [7-9], [18-24]. 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 which is then reviewed. The designers can

review their own design raising questions such as "Can this design be improved by making further changes?" Before finalizing their design, designers repeatedly examine their design and make changes, if necessary. Next, the design review may lead to new requirements analysis which may be revised again on the basis of a combination of software design reviews, new or changed requirements, or other factors which in turn may lead to the next software design. That is, the spiral process model [24], or an agile process [1, 4] is found to be a more productive software development process than the traditional processes. Certain aspects of software are such that after an initial analysis and assessment, iterative enhancements lead to significant progress in the development process. One of the major benefits of the iterative process is the improvements made in the design of user interfaces through successive iterations [26]. 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 shows an iterative process of design and review in the central core with solid bold 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. If needed, prototypes can be built and reviewed by stakeholders in order to enhance the design. The dotted arrows show other viable alternatives including iterations over the entire development process. User interface development requires adjustments and refinements that are best done in iterations [1], [3-5], [18-26]. Often defects are found during the review or evaluation process and these 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, as practiced in agile processes [1, 4]. The design review may be performed by the designers or by external reviewers, formally or informally. Designers often raise questions such as "Is this the best product design we can think of?" or "Is there a way to make the product better?" Answers to these questions often lead to iterative design.

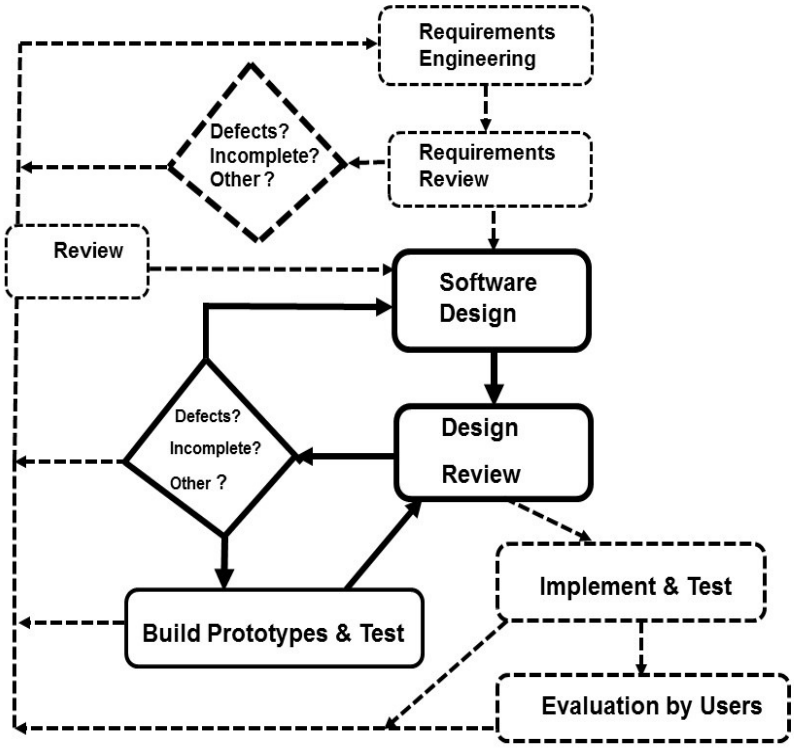

Figure 1: Iterative Design and Review

3. USE CASE VIEW

"Separation of concerns" is a fundamental premise of Software Engineering proposed by Dijkstra [33] 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 controlled [28]. The authors posit that the segmentation of the whole system into multiple views motivated by separation of concerns 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 in the operational software system. According to UML2.0, there are nine views for describing different aspects of software [9]. The views are: use case view, interaction view, state machine view, static view, design view, activity view, deployment view, model management view, and profile. A view is generally defined to be a subset of the UML modeling constructs representing certain aspects of the software [9]. Each view is thoroughly explained in [9] with one or more diagrams that visually illustrate 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 [8]. It is not appropriately used for user interface design in UML [9] although use cases have a lot to contribute to user interfaces. Use cases can clarify many important software issues early in the development process if they are adequately treated in the engineering process [3, 8, 9]. However, a very narrow definition of use case view is attempted in UML that basically ignores the nature and significance of use cases. "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" [9]. The perception of the outside agents and interactions mentioned above should be mediated through an interface such as a GUI, especially when the agents are humans. However, UML use case view fails to deal with user interfaces or interfaces between the actors and the use cases. In fact, there is no UML view that adequately deals with GUI features. 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 central problems with the UML use case diagram is that it totally ignores interfaces with the actors although each actor is shown to be using one or more use cases utilizing a line or association. Interactions among the actors cannot be shown in the same use case diagram. Each use case represent a service which can be illustrated in a UML sequence diagram [9]. For illustration purpose, consider a sample use case diagram shown in Figure 2.

The following initial requirements description characterizes the start of a small software project: *Develop a software system for computing areas of three types of play-place units: Rectangular, Circular and Triangular. A contractor in Los Angeles builds play-places (with materials such as wood, iron, pads, plastics etc.) at customer site using play place units of different dimensions. The charges are in dollars based on the area of each unit in square feet, plus the number of units. The software system is needed for computing the cost which is based on area. The cost is: $5.00 per square foot. Assume that users always use feet for entering the dimensions of the units. A Graphical User Interface (GUI) is required for user interactions. Additional typical assumptions can be made about this project.*

Most software projects start with some fuzzy requirements. Software engineers start their work with an initial requirements analysis. After performing the initial requirements analysis, software engineers may determine that the system must be web-based and should be available 24/7. The

access to the system is not required to be restricted with login ID. The system should be easy to maintain using web-based tools. The functional and nonfunctional requirements would be properly analyzed by the engineers. Finally, a software requirements specification (SRS) document would be prepared; it is generally use case driven [8]. The use case diagram for the play-place problem is given in Figure 2 in the standard UML notations [9].

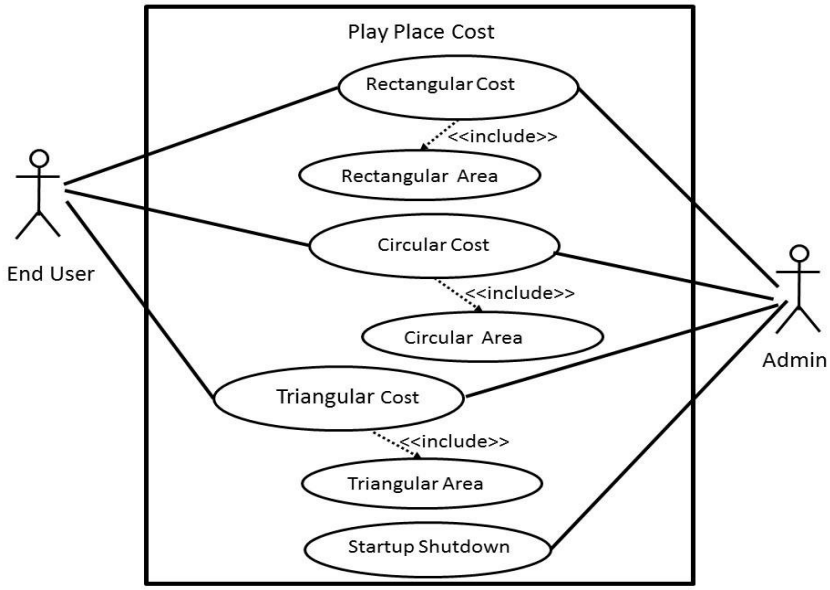

Figure 2: Use Case diagram in UML 2.0

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 [9] 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 depicted in a use case diagram. This research proposes that appropriate interfaces are included in augmented use case diagrams. Thus, the use case diagram given in Figure 3 is recommended for the sample software project mentioned above. It is important to note that the interfaces are shown with dotted rounded rectangles in Figure 3.

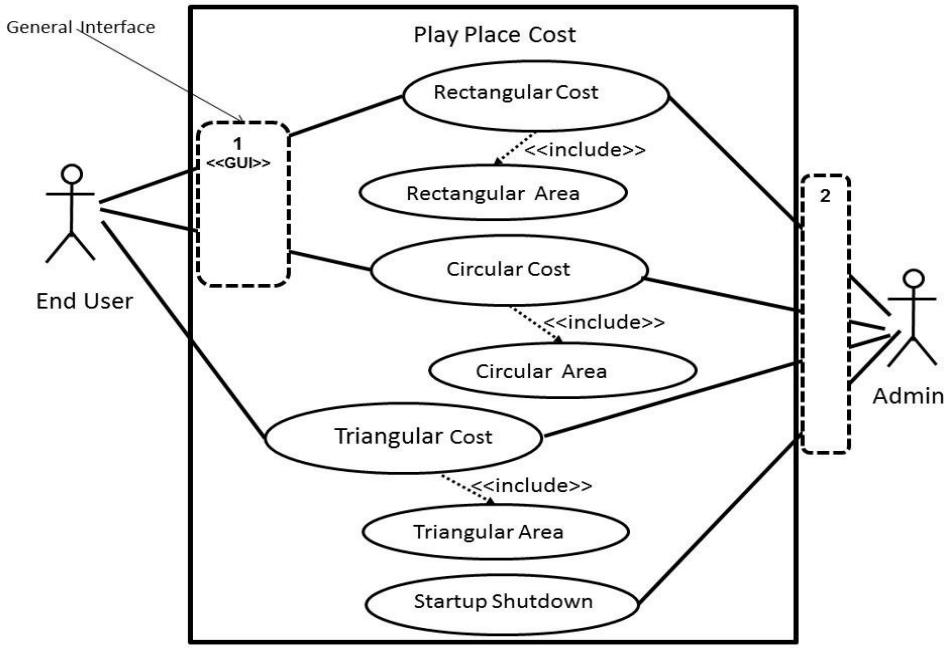

Figure 3. Augmented Use case diagram with general interfaces

These interfaces are referred to as the general interfaces in order to distinguish them from specialized interfaces such as provided interfaces and required interfaces mentioned in UML [9]. In order to refer to the general interfaces, they are sequentially numbered. If a general interface is to be developed as a part of the current software system, then it is shown within the system boundary; otherwise, it is shown outside the system boundary. As there are many different types of interfaces, some of them need to be marked for their importance. If an interface is a graphical user interface (GUI), then it is marked with the term <<GUI>> utilizing UML stereotypes [9]. 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 augmented use case view. When the actor is a human user, the general interface may be a GUI for appropriate interactions between the user and the system. For interactive systems, addition of GUIs to a use case diagram helps in understanding the perceived functionality of the system. 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 [9] 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. It is reasonable to be flexible about the notations of the general interface diagrams, especially if they are GUIs. 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. That is, we developed a prototype GUI using the Java programming language for the sample problem of play-place units

mentioned above in section 3 and took a screen shot of the GUI for Figure 4. It is reasonable to assume that through each subsequent iteration the GUI of Figure 4 will evolve and acquire better qualities as Figure 4 was taken from the 1st iteration.

The general interface diagrams such as the one shown in Figure 4 should be considered important for software design purposes. Jason Hong [1] asks an important question: "how do we effectively incorporate great design into products?" Currently, we cannot incorporate GUI design into standard UML based techniques. 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 perceived functionality is evidently perceived by the actors as the corresponding association links pass through the general interfaces in use case diagrams.

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" [5]. 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. Nobody should argue that interfaces are adequately treated in the UML design view and that augmentation of the use case view is not required, 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, 7, 27]. Separation of concerns [28] 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 its major goals without 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 [2-3]. 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 mentioned above. 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 [2-5].

Figure 4: General interface diagram

4. UI DESIGN PRINCIPLES

In this section, we propose a set of design principles for developing user interfaces. Jason Hong [27] 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 [27] 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 the design.

3. Utilize Object Oriented Design concepts throughout the development process.

4. Push the design-review-design cycle to its limits.
5. Consider separation of concerns in order to deal with all interactions among system elements.
6. Consider design principles as well as HCI data and user experience for innovative user interface solutions.
7. Include only those action features which are intuitively learnable; transform others to this category or to an automated category.
8. Maximize cohesion and minimize coupling among components.
9. Include error prevention and simple error handling.
10. 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 [27] with an example where contributions of subject matter experts are explained with an example of an experienced photographer. Experienced subject matter experts would be able to adequately explain what will, or will not, work in a given context.

Principle 3 suggests that object oriented design concepts need to be utilized throughout the iterative development process. Object oriented design elements such as buttons, windows, allow fast development cycles.

Principle 4 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 as shown in Figure 1. Through an iterative process designers may exhaustively explore many alternatives by critically examining their own designs and taking advantage of each iteration for improvements.

Principle 5 is based on a traditional strategy for dealing with complexity [2-4]. 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 [28].

Principle 6 is based on a commonsense integration of HCI factors [27-29], user experience, and other advanced design principles [27]. A good study of user groups helps in the understanding of user interface aspects which may stimulate innovative user interface constructs [29-32].

Principle 7 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 8 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 9 is based on Ben Shneiderman's suggestion [29] that when users are prone to make errors, an automated or easy recovery process should be used to prevent the error from occurring.

Principle 10 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 [9], [32-35] or the enhancements suggested above. The proposed design principles combined with augmented use case view have great potentials for smart user interface design. Steve Jobs and Jonathan Ive were committed to Apple's proclamations such as "Simplicity is the ultimate sophistication" [36]. They achieved simplicity by conquering complexities, not ignoring them [36]. Jobs and Ive forged a bond that led to "the greatest industrial design collaboration of their era" [36: Page 341]. The current industrial revolution is based on design contributions in the electronics and information technologies. Software engineers do not agree about a generally acceptable set of design principles and most of the principles are briefly presented in textbooks [4-5]. The principles discussed in this section are primarily based on modern best practices in the software industry; these best practices have supported innovation and growth [36]. It is difficult to learn user interface design from books in various environments [3]. However, user interface based interactive system design practices have improved in recent years [3, 36-38].

5. CONCLUDING REMARKS

Recent studies suggest that considerable progress has been made in user interface design practices. As user interfaces become increasingly important, a set of principles that direct selective iterative design techniques are considered helpful in developing an innovative approach towards user interface engineering. The set of principles proposed in this paper may provide sufficient clarity about the nature of innovations that are achievable through user interface engineering activities. It is reasonable to expect that various aspects of user interface modeling and design might be, procedurally, systematically reviewed and revised in an iterative evolutionary process that spans the entire lifecycle of a software system. In addition, 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 [32]. The enhancements suggested in this paper are most applicable in dealing with GUI aspects that are missing in the standard UML [10]. 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 using UML. The design techniques suggested above have the potential to help teachers and students in teaching-learning environments. Understanding the role of the teacher in the process of learning software design is a direction of future research for this field. The current best practices are developed primarily in the industrial environment with little or no participation from academia. Learnability of the best practices may be considered as an important part of future studies in this area.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the help and support received from the faculty members at National University, Department of Engineering and Computing, during the continuing research on this subject and the preparation of this paper.

REFERENCES

[1] Ousterhout, J. (2018) A Philosophy of Software Design, Yaknyam Press.

[2] Hong, J. (2010) "Why is Great Design so Hard?" Communications of the ACM, July 2010.

[3] Joo, H. (2017) A Study on Understanding of UI and UX, and Understanding of Design According to User Interface Change, International journal of Applied Engineering Research, Vol 12(20).

[4] Pressman, R. & Maxim, B. (2015) Software Engineering: A Practitioner's Approach, 8th edition, McGraw-Hill.

[5] Sommerville, I. (2010) Software Engineering, 9th Edition, Addison Wesley.

[6] Wang, Y. (2008) Software Engineering Foundations: A Software Science Perspective, Auerbach Publications.

[7] Shaw, M. & Garlan, D. (1995) "Formulations and Formalisms in Software Architectures", Computer Science Today: Recent Trends and Developments, Springer-Verlag LNCS, 1000, 307-323, 1995.

[8] Braude, E. & Bernstein, M. (2011) Software Engineering: Modern Approaches, (2nd Edition), John Wiley & Sons.

[9] Leffingwell, D. & Widrig, D. (2003) Managing Software Requirements: A Use Case Approach, Addison Wesley.

[10] Rumbaugh, R. Jacobson, I. & Booch, G. (2005) The Unified Modeling Language Reference Manual. (2nd Edition), Addison Wesley.

[11] Baniassad, E., Clements, P., Araujo, J., Moreira, A., Rashid, A. & Tekinerdogan, B. (2006) "Discovering Early Aspects," IEEE Software, 2006.

[12] Krechetov, I., Tekinerdogan, B. & Garcia, A. (2006) "Towards an integrated aspect-oriented modeling approach for software architecture design," 8th Aspect- Oriented Modeling Workshop, Aspect-Oriented Software Development (AOSD) 2006.

[13] Navasa, A. Pérez, M. A., Murillo, J. M. & Hernández, J. (2002) "Aspect Oriented Software Architecture: A Structural Perspective," Proceedings of the Aspect- Oriented Software Development (AOSD), 2002.

[14] Azevedo, J. L., Cunha, B. & Almeida, L. (2007) "Hierarchical Distributed Architectures for Autonomous Mobile Robots: A case study", Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation, 2007.

[15] Cerny, T., Cemus, K., Donahoo, M. & Song, E. (2013) "Aspect-driven, Data-reflective and Context-aware User Interfaces Design", ACM SIGAPP Applied Computing Review, volume 13(4), page 53-65, 2013.

[16] Knuth, D. E. (1969) Seminumerical Algorithms: The Art of Computer Programming 2. Addison-Wesley, Reading, Mass.

[17] Gries, D. (1981) The Science of Programming. Springer, 1981.

[18] Humphrey, W. (1989) Managing the Software Process, Reading, MA. Addison-Wesley.

[19] Pfleeger, S. & Atlee, J. (2010) Software Engineering, Prentice-Hall.

[20] Agarwal, B., Tayal, S. & Gupta, M. (2010) Software Engineering and Testing, Jones and Bartlet.

[21] Tsui, F. & Karam, O. (2011) Essentials of Software Engineering, 2nd Ed., Jones and Bartlet.

[22] Bass, L. Clements, P. & Kazman, R. (2003) Software Architecture in Practice, 2nd Edition, Addison-Wesley.

[23] Miller, J. & Mujerki, J. Editors, (2003) MDA Guide, Version 1, OMG Technical Report. Document OMG/200-05-01, http://www.omg.org/mda

[24] Boehm, B. (1986) "A Spiral Model of Software Development and enhancement," ACM SIGSOFT Software Engineering Notes, ACM, 11(4):14-24, 1986.

[25] Dey, P. P, Sinha, B. R., Amin, M. & Badkoobehi, H. (2012) "Augmenting Use Case View for Modeling", World Academy of Science, Engineering and Technology, Vol.6 (12), pages 1318-21.

[26] Nielsen, J. (1993) "Iterative User Interface Design," IEEE Computer vol.26 no.11 pp 32-41, 1993.

[27] Hong, J. (2010) "Why is Great Design so Hard (Part Two)?" Communications of the ACM, August 2010.

[28] Hursch, W. L. & Lopes, C. (1995) "Separation of Concerns", Technical Report, Northeastern University. 1995, Retrieved, July 11, 2014 from ftp://ftp.ccs.neu.edu/pub/people/lieber/crista/techrep95/separation.pdf

[29] Shneiderman, B., Plaisant, C., Cohen, M. & Jacobs, S. (2009) Designing the User Interface: Strategies for Effective Human-Computer Interaction (5th Edition), Prentice Hall.

[30] Tidwell, J. (2011) Designing Interfaces, O'Reilly, 2nd Edition.

[31] Nielsen, N. Gr. (2019) Why User Interviews fail? Retrieved June 14, 2019 from www.nngroup.com

[32] Loranger, H. (2014) UX Without User Research is not UX, retrieved April 14, 2015 from http://www.nngroup.com/articles/ux-without-user-research/

[33] Dijkstra, E. W. (1974) "On the role of scientific thought ", Retrieved August 15, 2015, from https://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html See also Effective Software Design, IASA Israel Meeting, retrieved April 12, 2019 from http://effectivesoftwaredesign.com/2012/02/05/separation-of-concerns/

[34] Agile Modelling, (2019) Introduction to the Diagrams of UML 2.X, retrieved April 14, 2019 from http://www.agilemodeling.com/essays/umlDiagrams.html

[35] Ambler, S. (2004) The Object Primer: Agile Model-Driven Development with UML 2, 3rd Edition. Cambridge University Press.

[36] Isaacson, W. (2011) Steve Jobs, Simon & Schuster.

[37] Glass, R. L., Vessey, I. & Ramesh, V. (2002), 'Research in software engineering: an analysis of the literature', Information & Software Technology 44(8), 491- 506.

[38] Hannay, J. E., Sjøberg, D. I. K. & Dyba, T. (2007), 'A systematic review of theory use in software engineering experiments', IEEE Trans. Software Eng. 33(2), 87–107.

[39] Jørgensen, M. & Shepperd, M. J. (2007), 'A systematic review of software development cost estimation studies', IEEE Trans. Software Eng. 33(1), 33–53.

Authors

Dr. Pradip Peter Dey is a Professor at National University, 3678 Aero Court, San Diego, CA, 92123, USA. He primarily teaches in the MS in Computer Science program, Department of Engineering and Computing. His major research interests are in Computational Models, Software Design, Mathematical Reasoning, Visualizations, User Interfaces and Computer Science Education. Phone: +1 (858) 309-3421; email: pdey@nu.edu.

Dr. Bhaskar Raj Sinha is a Professor at National University, 3678 Aero Court, San Diego, CA, 92123, USA. Dr. Sinha has more than 30 years of research and teaching experience in industry and academia. His major research interests are in Mathematical Reasoning, Digital Systems, Computer Architecture, Technology Management, and Engineering Education. Phone: +1 (858) 309-3431; email: bsinha@nu.edu.

Dr. Mohammad Amin is a Professor at National University, 3678 Aero Court, San Diego, CA, 92123, USA. He is the Academic Program Director for the Master's degree program for the MS in Electrical Engineering, Department of Engineering and Computing. His major research interests are in Computational Modelling, Wireless Communications, Relational Database, Sensors and Engineering Education. Phone: +1 (858) 309-3422; email: mamin@nu.edu.

Dr. Hassan Badkoobehi is with National University as a Professor in the Department of Engineering and Computing at 3678 Aero Court, San Diego, CA, 92123, USA. His major research interests are in Engineering Education, Environmental Engineering, Mathematics and Statistical Reasoning. Phone: +1 (858) 309-3437; email: hbadkoob@nu.edu.