World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:6, No:12, 2012

Augmenting Use Case View for Modeling

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

***Abstract*—**Mathematical, graphical and intuitive models are often constructed in the development process of computational systems. The Unified Modeling Language (UML) is one of the most popular modeling languages used by practicing software engineers. This paper critically examines UML models and suggests an augmented use case view with the addition of new constructs for modeling software. It also shows how a use case diagram can be enhanced. The improved modeling constructs are presented with examples for clarifying important design and implementation issues.

***Keywords*—**Software architecture, software design, Unified Modeling Language (UML), user interface.

I. Introduction

MODELS of computational systems are built for representing important aspects of a system in order to get a better understanding about design, experimentation and development. Modeling complex software systems presents formidable challenges [1]-[6] for all development phases including requirements engineering. High failure rates in software development projects are often attributed to difficulties with requirements engineering [7]. Emphasis on use case analysis and development has recently been steadily increasing because use cases clarify issues for complex software systems [7]. The Unified Modeling Language (UML) includes modeling of use case aspects in various views including the use case view [8]. This paper examines important development issues and the UML use case view followed by presentation of an augmented use case view. It suggests that certain interface elements should be properly included in use case diagrams. Representation of various software aspects are accomplished in visual diagrams of different views following the recommendations in the UML reference manual [8].

Modeling software aspects are based on best practices since practicing software engineers have developed useful strategies from their experience [1-12]. Software modeling is one of the most challenging tasks; it is partly scientific, partly intuitive, intricately multifaceted, highly creative, and deceptively flexible. Over the decades, several 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 [13]. David Gries [14] argued it to be a science. Watts Humphrey [15] viewed it as a process. In recent years, practitioners have come to realize that software is engineered [1]-[2], [4]-[20]. Engineering techniques have steadily improved the product quality in software development [1]-[2]. The role of user interface engineering has recently increased in most interactive software systems. User interface modeling and development presented additional challenges that are being addressed in an iterative process.

Pradip Peter Dey, Bhaskar Raj Sinha, Mohammad Amin and Hassan Badkoobehiare with National University, 3678 Aero Court, San Diego, CA 92123, USA. They are now with the School of Engineering, Technology and Media (phone: 858-309-3412; fax: 858-309-3420; e-mail: pdey@nu.edu; bsinha@nu.edu; mamin@nu.edu; hbadkoob@nu.edu).

II. Iterative Process

It is often suggested that software design is creatively built from requirements analysis in an iterative process [1]-[2], [4], [13]-[20]. In this process, after some initial requirements analysis, a software design representation is developed and then the requirements analysis is augmented on the basis of a combination of software design reviews, new or changed requirements or some other factors which in turn lead to a revised software design. That is, the iterative process of development or the spiral process model [21] is found to be one of the most productive software processes. Certain aspects of software are such that after an initial assessment, iterative refinements help. One of the greatest benefits of the iterative process is the improvements made in the development of user interfacesing each successive iteration [22]. The current study is based on the following iterative scheme where software design and modeling is followed by design review or evaluation.

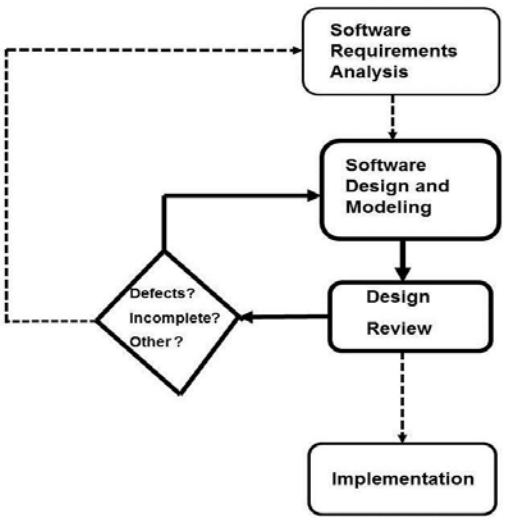

Fig. 1 Iterative Design and Modeling

The iterative process of design and modeling suggested in Fig. 1 allows developers to start with a highly abstract conceptual design and add details gradually in each successive iteration following the solid arrows. The dotted arrows show

other viable alternatives. User interface development requires adjustments and refinements that are best done in iterations[2], [22]. Often defects are found during the review or evaluation process and those defects need to be corrected. The design may start with just a few elements and other elements are incrementally added.

III. UML Views

Nine views are presented in UML 2.0 for describing different aspects of software [8]. These 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 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 visually.

The use case view is one of the most important views in the UML that presents use case features in a diagram called use case diagram. The use case view is well-motivated due to the role use cases play in defining requirements analysis and management [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 central problem 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. The use case view does not explicitly deal with user interfaces or interfaces between the actors and the use cases. The only diagram that characterizes the use case view is the use case diagram. This diagram 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. In order to illustrate the problem we present a sample case below.

Assume that a small software project started with the following initial requirements description: *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 studying the requirements, software engineers would discover that the system has to 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 by an administrator. The software engineers then would prepare a software requirement specification (SRS) document. A modern requirements analysis is generally use case driven [7]. A use case diagram is drawn with UML notations given in [8]. The use case diagram for the storage volume problem is given in Fig. 2 in the standard UML notations [8].

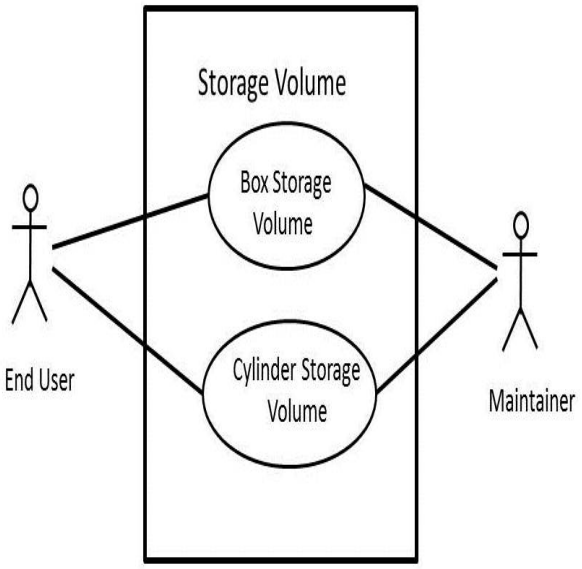

Fig. 2 Use Case diagram in UML 2.0

The problem with the UML use case diagrams such as the one given in Fig. 2 is that it ignores the interfaces between the actors and the use cases although it shows the actors as stick figures outside the current system boundary. For example, it shows the end user as an actor with two links to two use cases; but does not show any interfaces between them. Rumbaugh, Jacobson, 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 somewhere. We suggest that the use case diagram includes the appropriate interfaces. Thus, we suggest the use case diagram given in Figure 3 for the sample case mentioned above. Please note that the interfaces are shown with dotted rounded rectangles. We call such interfaces 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 referto the interfaces, they are numbered. If an interface is a graphical user interface (GUI) then we mark it 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 support the idea that user's perception about the functionality is modeled 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 emphasized in the UML modeling techniques leading to a degree of confusion for the development of modern interactive systems.

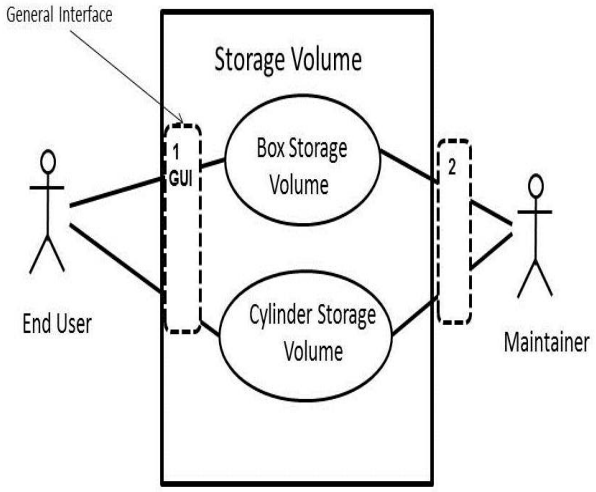

Fig. 3 Use case diagram with general interfaces

In addition to use case diagrams, the augmented use case view should have general interface diagrams. 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 Fig. 4 for the general interface 1 of Fig. 3.

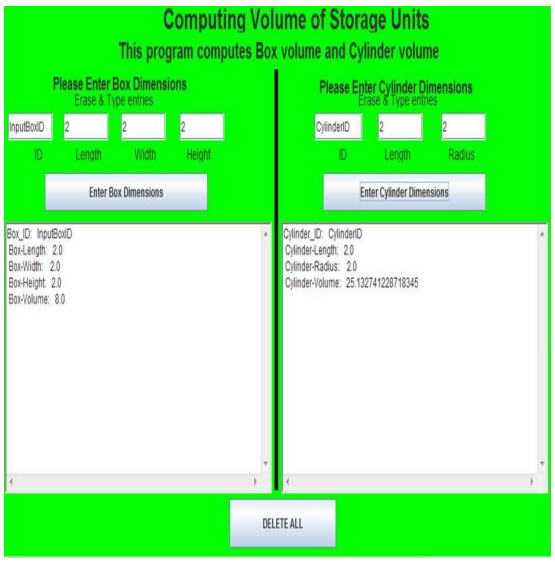

Fig. 4 General interface diagram

IV. Justifications

In this section, we provide logical justifications for the augmented use case view presented in the preceding sections. 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. 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 goes through the general interfaces. In addition, it includes general interface diagrams in order to add the elements of the general interface in some details. 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 allows paying attention to many aspects of user interfaces such as implications of user interface consistency: input mechanisms remain the same throughout the application.

One may argue that the UML design view treats interfaces appropriately; therefore, augmenting use case view is not required. This argument is not well-formed, because the design view just places the required and provided interfaces with the components. Extra emphasis is needed for interfaces of certain types, specially the GUIs. Modeling GUIs for interactive systems has become increasingly important in the past two decades [1], [2].

In addition, software engineering education with the UML requires guidance for learners which can be provided with the augmented use case view. Reasoning with the augmented use case view is better than that of traditional use case view, because the functionality of the system as perceived by the actors are more reasonable with the general interfaces. Exercises with the general interface constructs may also promote learning about user interfaces which is appropriate for educational environments.

V. Conclusion

Software engineers need a lot of help in their struggle against software complexity. Abstract models of software help in this struggle. With increased emphasis on user interfaces, it is reasonable that various aspects of modeling are periodically reviewed and revised. In this paper, the UML use case view is reviewed and suggestions are made for augmenting the use case view. The changes suggested here are not radical; they are in some sense additions to the traditional UML use case view. However, these additions may enhance software modeling significantly.

Acknowledgment

The authors gratefully acknowledge the help and/or encouragements received from John Cicero, Arun Datta, Gordon Romney, Ronald Gonzales, Alireza Farahani, and many others during the preparation of this paper and/or the research reported in it.

References

[1] R. S. Pressman, *Software Engineering: A Practitioner's Approach*. (7th ed.), McGraw-Hill, 2010.
[2] I. Sommerville, *Software Engineering*, 9th Edition, Addison Wesley, 2010.

[3] Y. Wang, *Software Engineering Foundations*: *A Software Science Perspective*, Auerbach Publications, 2008.
[4] M. Shaw, and D. Garlan, "Formulations and Formalisms in Software Architectures", *Computer Science Today: Recent Trends and Developments*, Springer-Verlag LNCS, 1000, 307-323, 1995.
[5] E. Braude, and M. Bernstein, *Software Engineering: Modern Approaches*, (2nd Edition), John Wiley & Sons, 2011.
[6] J. Hong, "Why is Great Design so Hard?", *Communications of the ACM*, July 2010.
[7] D. Leffingwell and D. Widrig, *Managing Software Requirements: A Use Case Approach*, Addison Wesley, 2003.
[8] R. Rumbaugh, I. Jacobson, and G. Booch, *The Unified Modeling Language Reference Manual*. (2nd Edition), Addison Wesley, 2005.
[9] E. Baniassad, P. Clements, J. Araujo, A. Moreira, A. Rashid, and B. Tekinerdogan, "Discovering Early Aspects," *IEEE Software*, 2006.
[10] I. Krechetov, B. Tekinerdogan, and A. Garcia,"Towards an integrated aspect-oriented modeling approach for software architecture design," 8th Aspect-Oriented Modeling Workshop,Aspect-Oriented Software Development (AOSD) 2006.
[11] A. Navasa , M. A. Pérez , J. M. Murillo, J. Hernández, "Aspect Oriented Software Architecture: A Structural Perspective," *Proceedings of the Aspect-Oriented Software Development* (AOSD), 2002.
[12] J. L. Azevedo, B. Cunha,andL. Almeida, "Hierarchical Distributed Architectures for Autonomous Mobile Robots: A case study",*Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, 2007.
[13] D. E. Knuth, *Seminumerical Algorithms: The Art of Computer Programming 2*. Addison-Wesley, Reading, Mass., 1969
[14] D. Gries, *The Science of Programming.* Springer, 1981.
[15] W. Humphrey, *Managing the Software Process*, Reading, MA. Addison-Wesley.
[16] S. Pfleeger, and J. Atlee, *Software Engineering*, Prentice-Hall, 2010.
[17] B. Agarwal, S. Tayal and M. Gupta, *Software Engineering and Testing*, Jones and Bartlet, 2010.
[18] F. Tsui, and O. Karam, *Essentials of Software Engineering*, 2nd Ed., Jones and Bartlet, 2011.
[19] L. Bass,P. Clements,and R. Kazman, *Software Architecture in Practice*, 2nd Edition Addison-Wesley, 2003.
[20] J. Miller, and J. Mujerki, Editors, MDA Guide, Version 1, OMG Technical Report. Document OMG/200-05-01, http://www.omg.com/mda, 2003.
[21] B. Boehm, "A Spiral Model of Software Development and enhancement," *ACM SIGSOFT Software Engineering Notes*, ACM, 11(4):14-24, 1986.
[22] J. Nielsen, "Iterative User Interface Design," *IEEE Computer vol.26 no.11 pp 32-41, 1993*

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

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, Technology and Media. 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; e-mail: bsinha@nu.edu).

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, Technology and Media. His major research interests are computational modeling, wireless communications, databases, sensors and engineering education. (phone: 858-309-3422; e-mail: mamin@nu.edu).

Hassan Badkoobehiis with National University as a Professor in the School of Engineering, Technology and Media 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; e-mail: hbadkoob@nu.edu).