Journal of Theoretical and Applied Information Technology
30th September 2011. Vol. 31 No.2

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

ASPECT-ORIENTED SOFTWARE DEVELOPMENT VERSUS OTHER DEVELOPMENT METHODS

[1]VAHDAT ABDELZAD, [2]FEREIDOON SHAMS ALIEE

[1] Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran.

[2] Electrical and Computer Engineering Faculty Shahid Beheshti University, GC, Evin, Tehran, Iran.

E-mail: v.abdelzad@srbiau.ac.ir, f_shams@bu.ac.ir

ABSTRACT

Nowadays, there are many software development methods that are used by developers in order to produce high-quality software systems. Aspect-oriented software development as a new software development method tries to obtain the quality of software systems through the modularity of crosscutting concerns in the whole of software development process. However, there are some doubts about using aspect-oriented software development instead of others in operational environments. Some questions that everyone asks are: why is aspect-oriented software development used? Why should we start to change our software development method? Therefore, in this paper to answer these questions, aspect-oriented software development is compared with use case driven development as one of the popular and proper methods for software development. The result of comparison is a set of technical and useful points that express clearly what improvements are obtained by using aspects in the process of software development.

Keywords: *Aspect-Oriented Software Development, Use Case Driven Development, Crosscutting Concerns, Aspect-Oriented Programming*

1. INTRODUCTION

The goal of developing a software system is satisfying stakeholders' concerns. A concern is one or several requirements depending on stakeholders and system development that is able to be implemented by a code structure. Concerns are dynamic and relative, that is, concerns relevant to a particular software unit which will change over time. The best method for producing a software system including many and various concerns is breaking a system into independent and loose coupling modules. These modules must be complied separately and also have interfaces for connecting to other modules [1].

In developing of software systems, separation of concerns has a critical role so software development methods have been trying to represent particular concepts and separation techniques to their development process. However, these concepts do not completely support all criteria, such as modularity. For example, structured methodologies suffered from a system state being controlled through a large number of global variables that could be modified by any line of code in the application. Furthermore, object-oriented methodologies have tangling and scattering problems in their structures.

Aspect-Oriented Software Development (AOSD) as a new development method tries to introduce aspect orientation for supporting separation of concerns in the best form so that it increases the software qualities such as modularity, maintainability, and evolution. In fact, it provides unique and advanced program structuring and modularization techniques [2]. However, there are some questions about its usability. The head question of all of them is what mechanisms and factors motivate us to utilize AOSD. In order to answer these types of questions, we preferred to depict them through comparing AOSD with Use Case Driven Development (UCDD). UCDD is one of the most practical and popular software development methods in operational and academic environment. Thus, results of comparing AOSD with UCDD will be easily understandable for

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

developers and researchers. We illustrate, in this comparison, differences which cause to move for using AOSD instead of UCDD.

The rest of this paper is organized as following. Section 2 gives brief information about AOSD, UCDD and their structure and development phases. Section 3 expresses benefits of AOSD versus UCDD. Section 4 presents technical and useful points for AOSD. Finally, Section 5 concludes this paper.

2. SOFTWARE DEVELOPMENT

The term software development is often used to refer to the activity of computer programming which is the process of writing and maintaining the source code, whereas the broader sense of the term includes all activities between the conceptions of the desired software and the final manifestation of the software. Therefore, the general phases of software development are composed of requirements engineering, architecture and design, implementation, and testing. AOSD and UCDD as two development method follow these phases and present their own specific approaches to each phase so that they can build a convenient software system.

2.1 Use Case Driven Development

In UCDD, use cases are primitive elements which whole development process is conducted by them. The concept of use case firstly was introduced by Jacobson and in his view each use case is the specification of a set of actions performed by a system, which yields an observable result that is typically valuable for one or more actors or other stakeholders of a system [3].

UCDD is started by use case elicitation. Use cases are elicited by some techniques from requirements of the system and then these use cases, in addition to the dependencies which are exist between them, are used for modeling of use cases. In fact, the system is modeled as black box in which each box represents a use case. After the system was modeled by use case, these use cases should be realized. In the process of realization, classes of use cases and their responsibilities are determined because the classes are used for realization of use cases. In other words, contents of black boxes are specified.

Finally, the realization of use cases, which is depicted as design diagrams, are verified to be certain that all stakeholders' concerns have been considered in the system design. After it was accepted that all stockholders' requirements have been satisfied, the classes which had been determined previously would be implemented by an object-oriented or structured language. At the end of this simple development process, the system is tested to be sure that software system supports all stakeholders' requirements without any mistake or misunderstand. Therefore in UCDD, use cases are a software engineering technique used to drive the whole software development life cycle [3].

2.2 Aspect-Oriented Software Development

AOSD as a new development method tries to develop a system through defining a new concept in order to get a high-modular system. This new concept is aspect and the aspect is structure that encapsulates crosscutting concerns. For the first time, aspect was introduced at the implementation level by Aspect-Oriented Programming (AOP) [4]. AOSD, now, is not just AOP because it encompasses a whole range of techniques to achieve better modularity. Therefore, AOSD focuses on modularity of crosscutting concerns in the whole of software development life cycle.

AOSD commences its process with a phase named aspect-oriented requirements engineering which focuses on separation of crosscutting functional and non-functional properties during requirements engineering which would otherwise be scattered across and tangled with other requirements [5, 6]. In other words, AORE approaches provide a representation of crosscutting concerns in requirements artifacts and focus on the composition principle [7]. AOSD is continued by aspect-oriented architecture which focuses on the localization and specification of crosscutting concerns in architecture designs. It improves and broadens the understanding of the identification and management of architecture design concerns. Aspect-oriented architecture utilizes aspect-oriented modeling and design for describing the architecture of the system.

Aspect-oriented design just lays in the fact that scattering and tangling in more traditional approaches can be modularized. Typically, such an approach includes both process and language. The process takes requirements as input and produces a design model. The produced design model represents separate concerns and their relationships. The language provides constructs that can describe the elements represented in the design and relationships which can exist between those elements.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

For implementation of a system based on aspect-oriented modeling, AOSD uses AOP that includes programming techniques and tools for modularity of crosscutting concerns in the code level [8]. In this process, independent and separate elements, which are determined separately, are implemented in the code level. Finally, AOSD utilizes aspect-oriented testing techniques for testing the system. The goal of testing is to show and review this fact that implemented aspects in the implementation level actually add crosscutting concerns characteristics to the system in order to addressing stakeholders' requirements [9].

According to described process, AOSD in the whole of software development life cycle focuses on two subjects which they are following:

- Concerns of a system, which cannot be designed separately, can be applied independently and separately to the system by defining new structures in design, modeling, and architecture.
- Increase understandability and maintainability by establishing models, designs, and implementations for components that have minimum overlap with each other.

3. BENEFITS OF AOSD VERSUS UCDD

We believe that the proper approach for examine a new software development is comparing it with an accepted software development method, because this approach is really obvious and understandable to majority of developers. Therefore, based upon this fact we express key points which can be discussed on these two methods.

1) Keeping peer concerns separate

Peer concerns are concerns that are distinct from each other. No peer is more important than another. However, significant overlap between peer concerns occurs when peer concerns are being implemented in the same system [10]. UCDD is unable to keep separate peer concerns on all of the development phases through requirements to implementation and testing. This deficiency of UCDD is a problem for system modularity. However, AOSD keep separate peer concerns. It used an aspect for each peer concern in order to implement them separately.

2) Keeping extension concerns separate

Extension concerns are concerns that are defined on top of a base concern. They represent additional service or features [10]. UCDD is unable to keep separate extension concerns as independent and distinct concerns, therefore, decreases understandability and maintainability of systems. However, AOSD in all of development phases can keep separate extension concerns. In AOSD, extensions are defined as advices then they are mixed with one another using composition mechanism.

3) Axis of development

Axis of development expresses what types of requirements will be used in the beginning of software development. The importance of this factor is that, if there are many developments axis for a development method then utilization of the development method increases. In the UCDD axis of development is functional requirement, because use cases are mainly a type of functional requirement. However in AOSD it is possible to consider both functional and non-functional requirements as axis of development. Indeed, concerns in the form of aspects are considered as fundamental development elements in AOSD.

4) Implementation language

In UCDD, an object-oriented language is used for implementing of systems. Object-oriented languages do not have ability for modularity of crosscutting concerns, so this causes tangling and scattering problems in the implementation phase. However, AOSD use aspect-oriented languages for implementation of systems. Aspect-oriented languages implement every crosscutting concern as local elements, so it does not create tangling and scattering problems.

5) Parallel development

UCDD focuses on use cases. Use cases are mainly a kind of crosscutting concerns which cannot be maintained separately, thus, it is impossible to develop a use case in parallel by different software development teams. However, AOSD can transform use cases to use-case slices and use-case modules. Whereof, a use-case module is independent so working on each use-case module can proceed in parallel as separate projects [10].

6) Full comprehension of modeling

UCDD is unable to comprehend all knowledge that exists during use case realization. This subject is due to cohesion of use cases and distributing their knowledge throughout models. In AOSD, extracting all embedded knowledge, which exists in use cases, can be obtained by defining aspects and other structures.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

7) System testing

For testing systems, UCDD use black-box, white-box, and stress tests [11]. In AOSD besides of such test, we have to perform other testing on aspect behaviors. Testing of aspect behaviors encompasses environment and type of its application into environment [12].

8) Selected operation for executing in each phase

In UCDD, the developer has to implement every operation in its own phase so it is impossible to implement the operations in other phases. However, in AOSD it is possible to perform the important subjects such as functional requirements in the early phases and postpone the other requirements to next phases. For example, security problem in UCDD is considered from the first development phase, but according to composition and decomposition capability in AOSD, it is possible that such operations (security) to be considered at the final development phase [10].

9) Interdisciplinary

This subject is due to applying development method in the systems that knowledge of software engineering may not satisfy their requirements (i.e. a system is very complex or systems of systems [13]). In UCDD, it is impossible that system development activities to be performed by people who are not familiar with software engineering techniques (they are not experts). However in AOSD, the system requirements that their answers are in other areas can be referred to experts in those areas. In this method, first, the system is developed by engineers, scientists, sociologists, etc through languages and techniques which they are familiar with them (e.g. UML) [14]. Second, an aspect-oriented mechanism combines their results together. So UCDD is not an interdisciplinary method.

10) Development nature

UCDD likes to implement system requirements that provide valuable results for users. However, AOSD satisfies mentioned objective and also it improves modularity of functional and non-functional requirements. In other words, UCDD does not have enough attention to modularity of non-functional requirements.

11) Separation of concerns

UCDD supports one-dimensional separation of concerns that is known as “tyranny the dominant decomposition” [15]. However, AOSD supports multi-dimensional separation of concerns [16].

4. TECHNICAL AND USEFULL POINTS

Aspect-oriented thinking as a method based on model has excellent characteristics. These characteristics are following: broad applicability, improved management of complexity, interdisciplinary, improved knowledge management, improved productivity, and reducing defects [14]. The result of aspect-oriented thinking in software engineering area is AOSD that includes all these characteristics (the characteristics have significant role in the success of AOSD).

Aspect-oriented programming which is used in AOSD supports modularity of crosscutting concerns in code level. Therefore, it prevents two important problems in UCDD known as scattering and tangling. Also aspect-oriented programming as a new language provides many application capabilities for developing of software systems in the various areas, such as middleware and real time systems [17].

AOSD with interdisciplinary characteristics in its own nature can be applied in development of software systems that software engineering knowledge is unable to solely satisfy them. As the thinking of the method is continuous learning and changing, we can first analyze, model and simulate a system with a certain method then determine requirement elements (such as hardware, software, and processors) for implementing and validating the system using the same method.

Process of AOSD by providing a method based on aspects keeps separate crosscutting concerns throughout software development life cycle. AOSD by identifying and separately modeling each use case (a type of crosscutting concern) in requirements and modeling phases causes that developers of next phases have better understanding of use cases. Moreover, they get familiar with non-functional requirements in the same modeling bases on aspects. Therefore, they can produce proper architecture and implementation. Aspect-oriented architect with obtaining proper feedback from aspects will be able to select proper architecture style and also produces an architecture based on aspect-oriented style which increases comprehensibility, usability and success of architecture. Finally, by applying aspects to development process of a system, we can provide well-defined and proper components. These components can encompass test cases in their own metadata when they are changing to off-the-shelf and business components [18].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

All characteristics explained in the above and previous section indicate that AOSD is an excellent software development method. However expressing these technical and useful points for acceptance of AOSD does not mean use cases have deficiencies because use cases can be considered as one of the important and basic concepts in AOSD and expressing these points about use cases only mentions the nature of use case driven development method.

5. CONCLUSIONS

In this paper, we compared two development methods namely aspect-oriented software development and use case driven development to answer these question: Why is aspect-oriented software development used? Why should we go to change their software development method? The result of the comparison is a set of benefits and differences that yield to a number of acceptance points. These acceptance points express that aspect-oriented software development with supporting aspect-oriented thinking and aspect-oriented programming has many capabilities than use case driven development. However, to persuade developers for using aspect-oriented software development we need to develop tools for each phases of it. Some tools have been developed but they are not complete. In the future, we are going to categorize these tools and create a framework for using them. In this framework, there will be some instructions for developers to follow them so that they product a high-quality software system.

REFRENCES:

[1] Grady Booch, Ivar Jacobson, and James Rumbaugh, Object Oriented Analysis and Design with Applications 3rd Edition, Addison Wesley, ISBN: 0-201-89551-X, 2007.

[2] Johan Brichau and Theo D'Hondt, Introduction to Aspect-Oriented Software Development, Report of the EU Network of Excellence on AOSD, 2005.

[3] Ivar Jacobson, Maria Ericsson, and Agneta Jacobson, The Object Advantage: Business Process Reengineering with Object Technology, Wokingham, England: Addison-Wesley, ISBN: 0-201-42289-1, 1994.

[4] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Christina Vidiera Lopes, Jean-Marc Loingtier, and John Irwin, "Aspect-Oriented Programming", European Conf. on Object-Oriented Programming (ECOOP), Springer, LNCS 1241, pp. 220-242, 1997.

[5] Awais Rashid, Ana Moreira, and Joao Araujo, "Modularisation and Composition of Aspectual Requirements", 2nd Int'l Conf. Aspect-Oriented Soft-ware Development, ACM, pp.11-20, 2003.

[6] Awais Rashid, "Aspect-Oriented Requirements Engineering: An Introduction", 16th IEEE International Requirements Engineering Conference, 2008.

[7] Ruzanna Chitchyan, Awais Rashid, Pete Sawyer, Alessandro Garcia, Survey of Analysis and Design Approaches, Report of the EU Network of Excellence on AOSD, 2005.

[8] Joseph D. Gradecki and Nicholas Lesiecki, Mastering AspectJ: Aspect-Oriented Programming in Java, Wiley Publishing, ISBN: 0-471-43104-4, 2003.

[9] Reza MeimandiParizi and Abdul AzimGhani, A Survey on Aspect-Oriented Testing Approaches, Fifth International Con-ference on Computational Science and Applications, IEEE, pages 78-85, 2007.

[10] Ivar Jacobson and Pan-Wei Ng, Aspect-Oriented oftware Development with Use cases, Addison-Wesley, ISBN: 0-321-26888-1, 2004.

[11] Roger s. Pressman, software engineering: A practitioner's Approach, fifth edition, McGraw-Hill, ISBN: 0-07-365578-3, 2001.

[12] Bart De Win, Wouter Joosen, and Frank Piessens, "Developing Secure Applications Through Aspect-Oriented Programming", 2002.

[13] Lisa Brownsword, David Fisher, Ed Morris, James Smith, and Patrick Kirwan, System-of-Systems Navigator: An Approach for Managing System-of-Systems Interoperability, Technical Note CMU/SEI-2006-TN-019.

[14] Shayne R. Flint, "Aspect-Oriented Thinking: An interdisciplinary approach to complex system engineering", college of engineering and computer science Australian national university, 2006.

[15] Peri Tarr, Harold Ossher, William Harrison, and Stanley Sutton, "N Degrees of Separation:

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Multi-Dimensional Separation of Concerns", ICSE, ACM, pp. 107, 1999.

[16] Ana Moreira, Awais Rashid, and Joao Araujo, Multi-Dimensional Separation of Concerns in Requirements Engineering, Int'l Conf. on Requirements Engg. (RE), IEEE CS, pp. 285-296, 2005.

[17] Omer Erdem Demir, Eric Wohlstadter, and Stefan Tai, "An Aspect-oriented Approach to Bypassing Middleware Layers", ACM, 2007.

[18] Sami Beydeda and Volker Gruhn (Eds.), Testing Commercial-off-the-Shelf Components and Systems, Springer, ISBN: 3-540-21871-8 2005.