Towards an Aspect-Oriented Agile Requirements Approach

João Araújo and João Carlos Ribeiro
CITI/Departamento de Informática
Faculdade de Ciências e Tecnologia
Universidade Nova de Lisboa
2829-516 Caparica, Portugal
ja@di.fct.unl.pt, joao.ribeiro@oni.pt

Abstract

The success of the application of agile software development approaches on building evolvable systems depends on how efficiently the changeable requirements are elicited and structured by software engineers. Current agile methods provide suitable approaches to define requirements of such systems in a systematic and simple way. Nevertheless, the crosscutting nature of some requirements is not dealt with by these approaches. Aspect-Oriented Requirements Engineering tackles the problem of crosscutting requirements, and its concepts can be used to address this problem in the context of agile software development. This work describes how aspects could be integrated to an agile software development approach at requirements level.

1. Introduction

Agile software development aims at fast communication and incremental delivering of software artifacts. Several approaches have been proposed and largely used in practice such as Extreme Programming (XP) [4], Crystal Methodologies [6], Scrum [18], Adaptive Software Development (ASD) [10], Feature-Driven Development (FDD) [15], Agile Modeling (AM) [2] and Dynamic Systems Development Method (DSDM) [20]. However, all these approaches have problems dealing with the crosscutting nature of some initial requirements. Crosscutting requirements are requirements that cut across other requirements and may result in scattered and tangled requirements descriptions. This problem may compromise the speed the system reacts to change. As a result, software evolution may be largely affected by crosscutting requirements (that are likely to change) when a traditional agile approach is adopted.

Therefore, it is essential that the crosscutting nature of some stakeholders' requirements be considered to avoid having the same requirements spread over the requirements descriptions, which turns to be more difficult to evolve and manage.

The aspect-oriented concepts can be used to answer this problem as crosscutting requirements can be seen as aspects. Aspect-Oriented Software Development [8, 12] promotes the separation of crosscutting concerns (aspects) during software development. Nevertheless, most research in this area has focused on the design and implementation phases of the software lifecycle. Aspect-Oriented Requirements Engineering (AORE) deals with crosscutting concerns at requirements level. Some approaches have been proposed including [3, 5, 14, 17, 22].

Our main aim is to develop a whole approach where aspect-orientation enriches agile software development in order to improve software evolution. In this paper, we focus on defining an aspect-oriented agile requirements approach where initial crosscutting requirements are modeled as coarse-grained scenarios (detailed scenarios can be further described, if desired). Scenarios are descriptions of desired or existing system behavior. Scenarios are commonly used in requirements engineering [1] because they are easily understood by all stakeholders. Scenarios may crosscut other scenarios. Crosscutting scenarios are called aspectual scenarios, as defined in [22]. Composition rules are defined to weave coarse-grained aspectual and non-aspectual scenarios to completely describe functionalities.

This paper is organized as follows. Section 2 gives an overview of the approach. Section 3 applies the approach to an example. Section 4 describes some

Proceedings of the 2005 Eighth International Workshop on Principles of Software Evolution (IWPSE'05)
1550-4077/05 $20.00 © 2005 IEEE

related work. Finally, Section 5 draws some conclusions and points directions to future work.

2. Overview of the approach

We start by identifying the main functionalities of the system. This is accomplished by discussing with the stakeholders, that must be members of the development team. The discussions are also used to identify coarse-grained scenarios for each functionality (from now on we call coarse-grained scenarios as just scenarios). By analyzing those scenarios, we observe some behaviors that crosscut several scenarios. These behaviors are abstracted as aspectual scenarios. Having the aspectual scenarios defined we must compose them with the non-aspectual ones they affect to have complete scenario descriptions. Compositions are specified through simple rules. They are used to identify ambiguities, errors, omissions and conflicts. These must be solved before implementation takes place. The final set of aspectual and non aspectual scenarios plus composition rules are used to implement users' functionalities. The process model is illustrated in Figure 1.

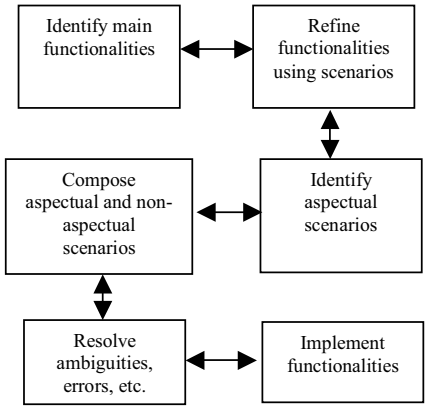

Figure 1. Process model.

3. Applying the approach to an example

We will illustrate our approach using a simple car parking example adapted from [22]. A brief description of the car parking system is given as follows:

"To use a car parking system, a client has to get a ticket from a machine after pressing a button. Then, the car is allowed to enter and park in an available place. The system has to control if the car parking is full or if it still has places left. When s/he wants to leave the parking place, s/he has to pay the ticket obtained in a paying machine. The amount depends on the time spent. After paying the client can leave by inserting the ticket in a machine that will open the gate for her/him to leave. Regular users of the parking system may pre-purchase time and enter/exit by inserting a card and PIN number which will result in money being deducted automatically from the user's account."

3.1 Identify main functionalities and refine them using scenarios

By analyzing the description above, we identify three main user functionalities: Enter Lot (UF1), Exit Lot (UF2) and Pay (UF3). Then we can refine each functionality into a number of scenarios. Table 1 shows a non-exhaustive list of them.

Table 1. Car parking scenarios

UF1-S1	Enter Lot, parking lot has space
UF1-S2	Enter Lot, parking lot has no space
UF1-S3	Enter Lot, regular user types in PIN and enters
UF1-S4	Enter Lot, regular user types in PIN; PIN incorrect
UF1-S5	Enter Lot, parking lot has space; machine is broken
UF2-S1	Exit Lot, driver inserts ticket; ticket paid
UF2-S2	Exit Lot, driver inserts ticket; ticket not paid
UF2-S3	Exit Lot, driver has no ticket
UF2-S4	Exit Lot, grace period from paying ticket exceeded
UF2-S5	Exit Lot, regular user types in PIN and exits
UF2-S6	Exit Lot, driver types in PIN; insufficient funds in account
UF2-S7	Exit Lot, driver inserts ticket; machine is broken
UF2-S8	Exit Lot, driver inserts ticket; ticket cannot be read
UF2-S9	Exit Lot, driver types in PIN; PIN incorrect
UF3-S1	Pay, driver inserts ticket, correct money inserted
UF3-S2	Pay, driver inserts ticket; ticket cannot be read
UF3-S3	Pay, driver inserts ticket; machine is broken
UF4-S4	Pay, driver adds money to PIN card

3.2 Identify aspectual scenarios

Among these, we observe that some scenarios are tangled. For example, handling error situations are tangled with normal ones. Also, these handling error situations crosscut several scenarios. This is the case of broken machinery, incorrect PIN etc. – all crosscutting. This leads to the refactored scenarios given in Tables 2 and 3, where the former shows non-aspectual ones and the latter shows aspectual ones.

Table 2. Non-Aspectual Scenarios

UF1-I1	Enter Lot, parking lot has space
UF1-I2	Enter Lot, parking lot has no space
UF1-I3	Enter Lot, regular user types in PIN and enters
UF2-I1	Exit Lot, ticket paid
UF2-I2	Exit Lot, ticket not paid
UF2-I3	Exit Lot, driver has no ticket
UF2-I4	Exit Lot, grace period from paying ticket exceeded
UF2-I5	Exit Lot, regular user types in PIN and exits
UF2-I6	Exit Lot, driver types in PIN; insufficient funds in account
UF3-I1	Pay, correct money inserted
UF3-I2	Pay, driver adds money to PIN card

For example, A1 is an aspectual scenario and when combined with the non-aspectual scenarios UF1-I1, UF2-I1, and UF3-I1, exceptional behavior is added to these scenarios.

Table 3. Aspectual Scenarios

A1	Machine is broken
A2	Ticket cannot be read
A3	PIN incorrect
A4	Driver insert ticket

3.3 Composing aspectual and non-aspectual scenario definitions

Having scenarios classified and separated into aspectual and non-aspectual, we still need to specify how they are combined together. To achieve that, we need to define simple composition rules. For example to compose scenario A1 with scenario UF1-I1 we have:

```
Compose A1 with UF1-I1
     where A1 XOR UF1-I1
```

That is, either the parking lot can be entered as there is plenty of space, or not, as the enter lot machine is broken. Since A1 and UF1-I1 are mutually exclusive, this is expressed using the exclusive-or operator.

We can also use a conjunction operator if we want both scenarios to happen. For example, if we want to compose A4 with UF2-I1 we have:

```
Compose A4 with UF2-I1
     where A4 AND UF2-I1
```

That is, the driver inserts a paid ticket and leaves the parking lot. These compositions can be more complex. For example, if we want to compose A1 with A4 and UF2-I1 we have:

```
Compose A1 with A4, UF2-I1
     where A1 XOR (A4 AND UF2-I1)
```

That is, either the driver exits the parking lot with a paid ticket or he cannot do that as the machine is broken. The final aim is to have complete composition expressions for each functionality, i.e., a functionality defined as a composition of aspectual and non-aspectual scenarios.

Note that requirements modularization is enhanced as scenarios and compositions are separated. Consequently, this facilitates and speeds system evolution as the impact of changes is localized. Many changes may even be related to compositions only.

3.4 Resolving problems and implementing

By analyzing the composition rules you may find ambiguities, errors omissions and conflicts in your scenarios. This analysis can be realized through inspections. Also, the participation of the stakeholders is crucial.

Conflict identification can be accomplished adapting the mechanism defined in [17]. The resolution of conflicts must be undertaken through negotiation. It is not the purpose of this paper to describe negotiation techniques. This will be considered in future work.

Finally, the agreed composition rules with the scenarios description can be used to guide the implementation. Here an aspect-oriented programming language could be used [8, 12].

4. Related Work

The aspect-oriented requirements engineering (AORE) approach, described in [17], proposes a model for aspect-oriented requirements engineering that supports separation of crosscutting properties at the requirements level. Requirements are defined using a viewpoint-oriented approach, PREView [19]. Composition rules are defined using XML. They use a list of constraint actions and operators, which are used to specify how an aspectual requirement influences or constrains the behavior of a set of non-aspectual requirements. They have a mechanism of conflict identification that can also be adapted to our approach.

In [14], functional requirements are specified using a use case [11] based approach, and quality attributes are described using templates, identifying those that crosscut functional use cases. Here, a set of UML models [21] are integrated to crosscutting quality attributes. [5] presents a model to handle separation of concerns during requirements. Here, they use the NFR framework catalogue [7] to help identify and specify concerns and a composition process with the introduction of match points, dominant aspects and composition rules using LOTOS operators. Baniassad and Clarke [3] propose Theme to provide support for aspect-oriented analysis and design. Theme supports the requirements analysis activity by providing an approach to identify base and crosscutting behaviors from a set of actions. An action is a potential *theme*, which is a collection of structures and behaviors that represent one feature. The results of analysis are mapped to UML models.

A closer approach to ours is depicted in [22], where composition of aspectual scenarios is presented. More explicitly, aspectual interactions are modeled using

Interaction Pattern Specifications [9, 13] and non-aspectual scenarios are modeled using sequence diagrams. Also, aspects and non-aspects are merged together before state machines are generated from them using the synthesis algorithm [23]. Then, these state machines are executed to validate the requirements against the stakeholders' needs. Our approach can be easily combined with this one if we want to refine our coarse-grained scenarios into detailed ones.

Nevertheless, none of the approaches discussed above has been instantiated to contemplate agile software development. An approach that considers that is described in [16]. Here, FDD is refined in order to incorporate aspect-oriented concepts.

5. Conclusions

This paper presented the description of an approach that incorporates aspect-oriented concepts into agile software development at requirements level. A process model was presented where basic functionalities were refined into scenarios, and crosscutting scenarios were identified from them. These so called aspectual scenarios are composed to describe the full behavior of the system's functionalities.

As a future work we aim to extend the composition rules to include more operators in order to define a complete composition language. Also we need to apply the approach to several case studies for validation purposes. Finally, we hope to develop a complete aspect-oriented agile software development approach and including some tool support to guarantee that the approach will be used in an agile fashion.

6. References

[1] I. Alexander and N. Maiden (eds.), *Scenarios, Stories, Use Cases: Through the Systems Development Life-Cycle*, John Wiley, 2004.

[2] S. Ambler, *Agile Modeling*, John Wiley, 2002.

[3] E. Baniassad, S. Clarke, "Theme: An approach for aspect-oriented analysis and design", *ICSE'04*, Scotland, May 2004.

[4] Beck, Kent, *Extreme Programming Explained: Embrace Change*, Addison Wesley, 2000.

[5] I. Brito, A. Moreira, "Integrating the NFR framework in a RE model", Early Aspects 2004, AOSD 2004, Lancaster, UK, March 2004.

[6] A. Cockburn, *Crystal Clear: A Human-Powered Methodology for Small Teams*, Addison-Wesley, 2004.

[7] L. Chung, B. Nixon, E. Yu and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer Academic Publishers, 2000.

[8] R. Filman, T. Elrad, S Clarke and M.Aksit (eds.), Aspect-Oriented Software Development, Addison-Wesley, 2004.

[9] R. France, D. Kim, S. Ghosh and E. Song, "A UML-Based Pattern Specification Technique", *IEEE Transactions on Software Engineering*, Vol. 30(3), 2004.

[10] J. Highsmith, *Adaptive Software Development*, Dorset House, New York, NY, 2000.

[11] I. Jacobson, Object-Oriented Software Engineering – a Use Case Driven Approach, Addison-Wesley, 1992.

[12] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier and J. Irwin. "Aspect-Oriented Programming", *ECOOP'97*, Vol. 1231, 1997.

[13] D. Kim, R. France, S. Ghosh and E. Song, "A UML-Based Metamodeling Language to Specify Design Patterns", *WiSME'03, with UML'03*, San Francisco, USA, 2003.

[14] A. Moreira, J.Araújo and I. Brito, "Crosscutting Quality Attributes for Requirements Engineering", *SEKE'02*, ACM Press, Ischia, Italy, July 2002.

[15] S. R. Palmer and J. M. Felsing, *A Practical Guide to Feature-Driven Development*, Addison-Wesley, 2002.

[16] J. Pang and L. Blair, "Refining Feature Driven Development - A methodology for early aspects", Early Aspects 2004, AOSD'04, Lancaster, UK, March 2004.

[17] A. Rashid, A. Moreira and J. Araújo, "Modularisation and Composition of Aspectual Requirements", AOSD'03, ACM Press, 2003.

[18] K. Schwaber and M. Beedle, *Scrum: Agile Software Development*, Prentice-Hall, 2002.

[19] I. Sommerville and P. Sawyer, *Requirements Engineering - A Good Practice Guide*, John Wiley, 1997.

[20] J. Stapleton, *Dynamic Systems Development Method - The method in practice*, Addison-Wesley, 1997.

[21] Unified Modeling Language Specification, version 2.0, 2005, In OMG, http://www.omg.org

[22] J. Whittle and J. Araújo, "Scenario Modeling with Aspects", *IEE Proceedings Software*, August 2004.

[23] J. Whittle and J. Schumann, "Generating Statechart Designs from Scenarios", *ICSE'00*, May 2000.