Volume : 1 | Issue : 5 | May 2012 ISSN - 2250-1991

Research Paper Information Technology

Identifying Crosscutting Concerns for Software Requirement Engineering

* Velayutham Pavanasam ** Chandrasekaran Subramaniam

* Research Scholar, Sathyabama University, Chennai, Tamil Nadu, India

** Professor, Kumaraguru College of Technology, Coimbatore, Tamil Nadu, India

ABSTRACT

Viewpoints, goals and use cases are the various methods through which the separation of stakeholders concerns can be achieved. Many techniques have been developed for crosscutting concern identification. Crosscutting concerns enhances the corrections of the software design, reduces software complexity, saves cost, improves traceability among requirements, avoids coupling between requirements and eases function modularization. There is a large impact of crosscutting concerns during the early stages of software development. Unique requirements are identified and structured into concerns such as use cases and viewpoints. The various steps in crosscutting concerns have been discussed in this work.

Keywords : Concerns; Crosscutting; Requirement Engineering; Viewpoints; Functional Requirements

Introduction

Requirement engineering involves information collection and structuring tasks which can be categorized into requirement gathering and requirement analysis. Output of the requirement gathering will be the requirement documents. These documents will be analyzed in the requirement analysis stage. Unique requirements are identified and structured into concerns. Examples include use cases, goals and viewpoints. The documents will be signed by the stakeholders during requirement analysis stage known as the specification document [1]. Aspect-oriented software development advocates the separation of crosscutting concerns during the software development. Most research in this area has focused only on the design and implementation phases of the software lifecycle. The early aspects initiative refers to crosscutting properties at the requirement and architecture level [2]. Aspect provide a structuring construct that allows program code to be written or re-written to facilitate the representation of multiple concerns and to alleviate tangling of overlapping, aka cross-cutting concerns [3]. Separation of concerns refers to the ability of identifying, encapsulating and manipulating parts of the software that are crucial to a particular purpose. Separation of concerns reduces the software complexity and enhances understandability and traceability throughout the software development process. It also minimizes the impact of change promoting evolution [4]. Problems caused by crosscutting concerns can be observed in legacy software systems in particular, either because legacy system cannot be easily changed due to their design or because old programming language lack the features required. The programmers are then forced to use an idiom for the implementation of a concern at each and every location in the source code [5]. Automated techniques apply their rules consistently, but they might not find the concerns that the developer is interested in. Execution trace-based techniques, for instance, miss concerns that cannot be isolated by a given test run. These techniques miss nonfunctional concerns, such as logging and error handling. Aspect mining and static analysis techniques are useful at generating suggestions for possible concerns, but human interpretation is still required. Another level of inconsistency is introduced when concerns are assigned to code because existing guidelines are ambiguous. These inconsistencies ensure that experimental results are not repeatable and lead to misguided assessments of the nature and extent of crosscutting in the program [6]. Use cases are the main structure in requirements engineering of RUP and many others software engineering process definitions. They describe several requirements in only one use case, i.e., it is a composite requirement. A use case comprises the behavioral requirements in its main structure and also contains sections for others types of requirements specification, such as business rules and data input. The RUP style use cases templates define a section named special requirements as the place to put this kind of requirement. Use case sections and characteristics are decomposed in requirements types. These types are then qualified and classified as representing or not a crosscutting concern. AICC-UC concern approach is based on the notion that the base concerns are primary actor's main goals on each use case. Use case must cover interests of primary actor and protect interests of all others stakeholders. Functional or nonfunctional requirements that are not directly related to the accomplishment of main goal crosscut the base concern to protect the interest of stakeholders in the use case [7]. A tool, 3CI that automatically identifies crosscutting concerns and their relationships at the requirement level is described. The tool utilizes NLP techniques to extract linguistic properties and exploits the properties to identify crosscutting concerns and its influences in a requirements document. NLP techniques such as part-of-speech analysis, word frequency analysis and dominant verb analysis contribute in the processing of software requirements phrases to assist aspects mining [8].

Identifying Crosscutting Concerns

According to Wikipedia, cross-cutting concerns are aspects of a program which affect other concerns. These concerns often cannot be cleanly decomposed from the rest of the system in both the design and implementation, and can result in either scattering (code duplication), tangling (significant dependencies between systems), or both. Consider an application is written for handling medical records, the bookkeeping and indexing of such records is a core concern, while logging a history of changes to the record database or user database, or an authentication system would be cross-cutting concerns since they touch more parts of the program. A requirement in requirement specification document represents a concern. The separation of concerns has been emphasized by software engineering principles for better understanding of concerns. Object oriented paradigm supports the modularization of concern. Improper implementation of the feature and the limitations of programming language construct leads to scattered and tangled implementation of the concern.

Every concern might not get modularized into a separate module. Such concerns whose implementation is scattered over more than one module are called crosscutting concerns. Such concerns lead to tangling of code. Empirical studies have revealed that scattered and tangled code degrades the code quality. The negative impact of scattered and tangled code is reflected not only by internal quality metrics but also by the external quality metrics. Poor modularization of crosscutting concerns results in source code which has more defects and is difficult to maintain. For development of good quality software, it is essential to identify crosscutting concerns. Identification of crosscutting concerns at different stages of software development is essential for 3 reasons. First, for refactoring of legacy system to aspect oriented system. Secondly, for modularized implementation of concern, its crosscutting nature needs to be identified at analysis and design level. Thirdly, for appropriate distribution of testing effort error prone crosscutting code needs to be identified [9].

Software systems are bound to suffer from a phenomenon known as crosscutting concerns. Crosscutting concerns are concerns – pieces of functionality – that cannot fit neatly into a system's design. Example of crosscutting concerns is error or exception handling. Since each module deals with exceptional situations, code for error-handling is scattered across system. When stakeholder's requirements are represented by viewpoints, the overlapping requirement denotes overlapping crosscutting requirements. They provide useful input into aspect-oriented design and implementation and manage any inconsistency which arises during overlapping. When identification and documentation of crosscutting requirements and influences are conceivable, various stages of the development and maintenance processes are:

- W*hile requirements modeling*: After requirements have been elicited they should be modeled. Modeling is a highly analytic activity whose goals are for example to structure the requirements (normally top-down), to identify and model dependencies and other relations among them, to identify and eliminate inconsistencies and to identify and clarify ambiguities and vagueness. It would of course also be natural and highly desirable to identify and model crosscutting requirements and influences already at this stage.

- W*hile writing the requirements specification*: Writing a specification normally follows requirements modeling. However, it is often the case that a specification is written directly after requirements have been elicited thus bypassing the modeling stage. In this case writing the specification is a highly analytic process as well. It should then also include identifying and documenting crosscutting requirements and influences.

- A*fter writing the requirements specification*: Although it is best to identify and document crosscutting requirements and influences when they arise, i.e. during requirements analysis, we have to face the fact that countless requirements specifications exist, written without the aspect of crosscutting requirements in mind. With hindsight however, it may become desirable for developers to identify them in such documents. It then becomes necessary to "mine" crosscutting requirements and influences from them.

- *During downstream activities*: The crosscutting nature of some requirements and their influences may and will of course also be detected during activities later in development or maintenance and should then be documented [10].

The framework of aspect-oriented software reverse engineering is proposed for the solution of comprehension and evolution problems of crosscutting properties in legacy system. Based on it, an approach of use-cases driven formal concept analysis is discussed. The goal of this approach is recovering system's crosscutting concerns on requirements level. Execution profiles of legacy system are analyzed using concept lattices and the invoked computational units that traverse system's use-case models can be identified with this approach. Finally, they can be abstracted into early-aspects for re-engineering of the legacy systems with AOSD (Aspect-Oriented Software Development). Compared with existing aspect-mining techniques, which are mainly applied for refactoring legacy system's program source codes, this given approach is more effective when it is used for comprehending and evolving legacy system on higher abstract level. The result shows that quite a few crosscutting properties of legacy system can be recovered at requirements level with the introduced domain knowledge with the help of a case study [11].

Steps in Crosscutting Concerns

Various steps in crosscutting requirements are:

· Identify and specify functional requirements
· Identify and specify crosscutting concerns
· Integrate functional requirements with crosscutting concerns

Step 1: *Identify and specify functional requirements*: Requirements can be divided into functional and non-functional requirements. It is important to identify the use cases and actors. When a use case occurs, a template can be created describing the use case in a detailed manner.

Step2: *Identify and specify crosscutting concerns*: Non-functional requirements are global properties crosscutting concerns that can influence part or the whole system. Crosscutting concern can be mapped into a function and aspect. Extensible Markup Language (XML) can be used to specify the crosscutting concerns and requirements. An iterative and incremental approach must be followed to fill in the information available in the template.

Step3: *Integrate functional requirements with crosscutting concerns*: The complete system can be obtained by integrating the functional requirements with crosscutting concerns. Unified Modeling Language (UML) diagrams can be used to perform this integration efficiently. UML includes a set of graphic notation techniques to create visual models of software-intensive systems. Use case diagram includes a new case during higher abstraction level and makes the crosscut initial use cases.

These steps can be represented as shown in figure.1.

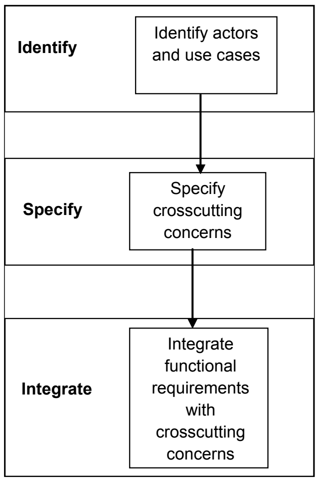
Identify
Identify actors and use cases
Specify
Specify crosscutting concerns
Integrate
Integrate functional requirements with crosscutting concerns

Fig. 1. Steps in Crosscutting Concerns

Conclusion

It is essential to apply the separation of crosscutting concerns during all stages of software development through which reusability, maintainability and comprehensibility improves. Even though large numbers of existing methods can be used for identifying crosscutting concerns, it is found that none of these methods can automatically identify crosscutting concern. Both functional non-functional requirements must be identified and their elicitation must be accurate and complete. Steps in crosscutting concerns have been proposed so that separation of concerns at the requirements level can be achieved. Users can easily identify and manage crosscutting concerns for software requirement engineering.

REFERENCES

B.S. Ali & M.Z. Kasirun, "A Review on Approaches for Identifying Crosscutting Concerns", IEEE International Conference on Advanced Computer Theory and Engineering (ICACTE), pp. 855-859, 2008. | Abdelkrim Amirat, "Towards a Requirements Model for Crosscutting Concerns", Information Technology Journal, Vol. 6(3), pp. 332-337, 2007.| Bashar Nuseibeh , "Crosscutting Concerns", 3rd International Conference on Aspect-Oriented Software Development (AOSD'04), pp. 3-4, 2004. | A. Moreira, J. Araujo & I. Brito, "Crosscutting Quality Attributes for Requirements Engineering", 14th International Conference on Software Engineering and Knowledge Engineering (SEKE'02), pp. 167-174, 2002. | Magiel Bruntink, "Analysis and Transformation of Idiomatic Crosscutting Concerns in Legacy Software Systems", IEEE 23rd International Conference on Software Maintenance (ICSM), pp. 499-500, 2007. | M. Eaddy, A. Aho & C. Murphy, "Identifying, Assigning and Quantifying Crosscutting Concerns", First International Workshop on Assessment of Contemporary Modularization Techniques (ACoM'07), pp. 1–6, 2007. | P. Pretes, F. Costa, F. Silveria, et al. "AICC-UC An Approach to Identify Crosscutting Concerns Based on Use Cases", Latin America Workshop on Aspect-Oriented Software Development (LA-WASP), 2011. | B.S. Ali & M.Z. Kasirun, "A Review on Approaches for Identifying Crosscutting Concerns", IEEE International Conference on Computational Intelligence for Modelling Control and Automation, pp. 351-355, 2008. | A. Kaur & K. Johari, "Identification of Crosscutting Concerns: A Survey", International Journal of Engineering Science and Technology, Vol. 1(3), pp. 166-172, 2009. | Lars Rosenhainer, "Identifying Crosscutting Concerns in Requirements Specification, 2004. | S. Yang & P. Zhong,"Recovering Crosscutting Concern from Legacy Software Based on Use-Cases Driven Formal Concept Analysis", International Conference on Computational Intelligence and Software Engineering (CiSE), pp. 1-4, 2010