ISSN 2229-5518

An Early Approach to Identify and Classify Crosscutting Concerns in Aspect-Oriented Requirement Engineering (AORE) for Better Software Modularity

Waseem Baig, Dr. Muhammad Ahsan Latif, Ahmed Mateen, Tasleem Mustafa and M. Imran Habib

University of Agriculture Faisalabad Pakistan

Abstract

In this era of science and technology, where dependability on computer based systems is increasing, the size of software code is also increasing due to diverse nature of user requirements. Large software systems must have understandable code and impact of change should also be known otherwise it would be quite reasonable to say that software is very complex. The software complexity can be minimized by increasing the modularity either by using Procedural languages or Object-oriented languages but Code-tangling and Code-scattering cannot be avoided entirely by these approaches. Code-tangling and code-scattering would ultimately result in poor-traceability and difficulty in software evolution. Requirement gathering is a key task for any project but for a successful completion of any project common functionality of all the modules should also be known which is called crosscutting concerns. Besides identification of crosscutting concerns, the classification of crosscutting concerns is also very important especially when software modularity is concerned. Without identification and classification of crosscutting concerns, software development process would be devastating and is simply wastage of both time and money. The better identification, classification and separation of crosscutting concerns mean better modularity of the software system thus results in enhanced software quality. Aspect-oriented software engineering is relatively a new software paradigm which actually deals with the crosscutting concerns to avoid code tangling and scattering for better software modularity. The purpose of this paper is to establish a roadmap for identifying and classifying crosscutting concerns for better software modularity to support Aspect-oriented software development approach.

Index terms – Approach, Aspect-oriented, Classify, Code-tangling, Code-scattering, Crosscutting, Concerns, Engineering, Identify, Modularity, Requirements.

1 Introduction

Software modularization faces Crosscutting concerns as a major problem which may create a hurdle for many of the upcoming needs and a reason for the failure of the software as well. Although AOP (Aspect Oriented Programming) [1] provide a good solution to problems, which may arise from AOP and can affect Object Oriented implementation after encapsulation in a crosscutting-way [2]. Need of identification of such aspects at early stage of software life cycle is one of the most significant issue [3].

If we identify the aspects at requirement level, the development will have the ability of evolution and dependability [4, 5, 6]. A lot of research has already been done with the provision of different kinds of methods to identify the aspects at early stages of software life cycle, but limited practical output is given against much effort and a lot of previous knowledge which sometimes create a barrier between the theory of such methods and the practical approach to implement these methods. Also, software engineers are not bound to furnish these methods before use to make them in accordance with the software development processes in common. To lessen the effect of AOP adaptation, it is necessary to chose best practices and principles of Aspect Oriented Software Development with common software development methods.

Our paper presents a new approach to classify the aspects whether they are crosscutting concerns or not. Our approach will work at use case model level of software life cycle, which is a requirement gathering level approach though with all salient feature of software development but without introducing new models or concepts to save the developers' community from any suffering or chaotic study.

The organization of paper is as follows: Section # 2 is about an introduction to early aspects nature and use cases. Section # 3 presents our approach using use case models to identify crosscutting concerns. In section # 4, a hypothetical company is chosen as a case study to present the application of our approach with an evaluation based on Concern-Morph which is a plug-in of Eclipse Integrated Development Environment. Section # 5 is about related work. At the end, section # 6 presents conclusion, issues and motivation for future work.

2 Background

To support modeling and better implementation of AOP development, there must be some predictive software methods to identify the aspects in software life cycle. By using requirement models definition and its analysis can help to demonstrate the crosscutting nature of certain aspects.

Use cases are used to set an interaction scenario within a system. The very good nature of use cases help to use them in different software development approaches [7]. Uses of case are used to describe the interaction of a user with the software system. Mainly use cases are used to describe functional requirements of the software system, which actually consists of behavioral requirements of the system under design though case cases can be used to describe non-functional requirements of the system under study as well [8].

Use of use cases is not only a technique which is used at requirement gathering level but it's a technique which is smoothly helpful in the all stages of software life cycle. In this paper, we have used use cases only for a requirement gathering technique which is presented by the specification and diagrams of use cases. The use case specification and diagrams collectively comprises of various views of entire specification of a software system [8].

3 Methodology

In this section, the proposed technique is presented in which use cases description is used to identify the crosscutting concerns. RUP requirement engineering and many other software requirement engineering processes are using use cases their main structure [8]. A use case is actually a composition of different requirements as it may contain several requirements inside it to show their responsibilities and objectives. The main structure of use case is used to describe the functional requirements of the software under design but it also has sections to show other types of requirements such as non-functional software requirements, data inputs as well as business rules. A "special requirement" section is included in use case templates of RUP style to describe non-functional or behavioral requirements of the software system under design. After designing a system using use cases, different sections of use case are then interpreted into relevant functional or non-functional requirements. Furthermore, this is the stage to then decompose these use cases to identify and classify crosscutting concerns [7].

Our approach takes primary actors main goals as base concerns in each of the use cases. Primary actors' interests must be protected by every use case as well as protection of interests of all the stakeholders in system under design [7]. So it is justifiable that all type of requirements either functional or non-functional requirements that are not part of the objectives of main goal of the system under design crosscuts the concerns at base while protecting the interest of all the stakeholders in that use case model.

Table # 1 consists of decomposed requirements with indication whether a crosscutting concern is represented at this stage or not as well as the identification of crosscutting concerns.

Rule ID	Requirement's type	Whether crosscutting concern or not?	Found at level?
BR01	Main Goal	No	Short description/ contained in a certain section
Justification: This requirement described summarized information of primary actors' main goal and does not covers any functional or non functional behavior of the system under design.			
BR02	Basic flow	No	Specification of the use case
Justification: This requirement describes the behavior of the system under design. This is a base concern whose addressed concern is the primary actors' main goal.			
BR03	Alternative flow	Yes (if it is not the basic flow)	Specification of the use case
Justification: Alternative flow also has a goal and guarantees [7, 9]. The alternative flows ultimately have to interact with the basic flow with its goal. Aspects can be used as an alternative flow [8]. Also, initial aspects can be depicted from linked requirements [5]. In some cases many alternative flows can be used for a basic flow and all the alternative flows will meet the goals of the basic flow as well. If alternative flows are going towards base flows then they could also be taken as basic ones [7]. As these basic concern of the primary actors is being addressed by these alternative flows so they are not crosscutting concerns.			

BR04	Extension use case	Yes	Diagram of use case
Justification: same as alternative flows, these are extensions to use cases and with the help of these extensions we can add new behavior to the existing use cases [8]. These extensions have nothing to do with the main goal of the system under design and whatsoever is the case, coupling nature of these extensions we can categorize them as crosscutting concern.			

BR05	Inclusion use case	Yes (if and only if more than one use cases includes them)	Includes use case diagrams, hyperlinks and references that are made primary use case
Justification: The basic requirement is modularization for which separation of crosscutting concerns is necessary. This type of use cases point towards reusability or behaviors' reuse. Actually when a use case is being referred by more than one use cases then we can say that its behavior crosscuts. If this relation is one to one then this will not be considered as crosscutting and may be included for formal process, only.			

BR06	Pre-condition	No	Specification of use case
Justification: These types of requirements are out of the boundary of implementation of use case itself [10]. As these are outside the systems' boundary so they are not considered as implemented concerns.			

BR07	Post-condition	Yes (if and only when present in at least two use cases with same requirements)	Specification of use case
Justification: if post-condition is being reproduced by more than one use case then it can be considered as a sub-goal for all the relevant use cases, so it addresses the same requirement and also crosscuts the main goal for all the relevant use cases.			

BR08	Business rule	Yes	Alternative flows and certain special requirements in the specification of use case
Justification: As business structure is identified by business rules so functional or behavioral requirements are not described by them and same alike structure of the business, they do affect some functionality of the system under design but not a representation of the system under design [5, 11, 12]. Many development approaches indicate that business rules are candidates of aspects and certainly purposes solutions by using aspects.			

BR09	Non-functional requirements	Yes	Special requirements
Justification: The major category of potential crosscutting concerns is non-functional requirements and also broadly referenced by many researchers [5]. Simply, if functional or behavioral requirement is the main goal of a use case then if it is not important for the success of a use case, non-functional requirement will crosscut.			

Table - 1

4 Case Study

A hypothetical company's HR software requirement specs are produced for the evaluation of our approach and then an implementation plan is also developed to show and discuss the results for our conclusion.

5 Analysis

The main goal of the system under study is to manage the employees; hiring, firing, assigning departments to the newly hired employees, daily "IN" and "OUT" of the employees and then calculating the wages/ salary of the employees.

The main steps of the process are as follow: (i) a person should be hired through standard operating procedure (SOP); vacancies are published through company's website (ii) A person should register through website (iii) after proper interview, employees join the company and then his/ her card is get printed and proper "IN" attendance is marks through the system under study (iii) if the employee left the company, wages/ salary of the employee is calculated (iv) if a person tries to register for an already filled post, he is informed that the post has already been filled and this that he may register himself/ herself for future vacancies and the person will be intimated accordingly upon the availability of a vacancy. Figure#1 is representing the use case diagram, which shows the company's HR system in a broader sense. Furthermore, table#2 is created with identified crosscutting concerns for the said system which is going to use the approach.

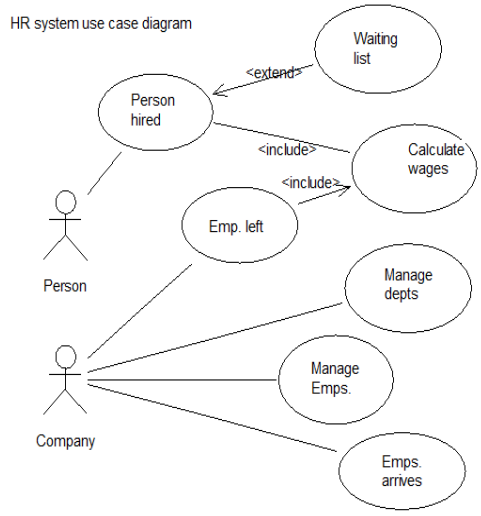

Figure # 1

ID	System Requirement	Use case	Rule ID
CCC01	To handle waiting list of candidates	Use case diagram	BR04
CCC02	To calculate wages/ salary	Use case diagram	BR05
CCC03	Do not allow application against filled vacancy	Hire employees use case specs	BR03
CCC04	Cancel application against filled vacancy	Hire employees use case specs	BR03
CCC05	Management of existing employees	Hire employees use case specs	BR03
CCC06	Filled posts	Hire employees use case specs	BR03
CCC07	Candidate is not register with waiting list	Handle waiting list use case specs	BR03
CCC08	No wages/ salary	Calculate wages/ salary use case specs	BR03
CCC09	Candidates in the waiting list	Employees left use case specs	BR03
CCC10	Employee left without intimation	Employees left use case specs	BR03
CCC11	Delete employee	Manage departments use case specs	BR03
CCC12	Employee department cannot be changes	Manage departments use case specs	BR08
CCC13	Department cannot be deleted	Manage departments use case specs	BR08
CCC14	Employee number missing	Employees "IN" use case specs	BR03
CCC15	Change employees department	Employees "IN" use case specs	BR03
CCC16	Employees' data not found	Employees "IN" use case specs	BR03
CCC17	Delete blank data employees	Manage employees use case specs	BR03
CCC18	Working employees cannot be deleted	Employees "IN" use case specs	BR08
CCC19	Wages per day	Calculate wages/ salary use case specs	BR08
CCC20	Employees categories	Calculate wages/ salary use case specs	BR08
CCC21	Bonus and increments	Calculate wages/ salary use case specs	BR08
CCC22	Messages and alerts sent	Employee hired and left use case specs	BR07
CCC23	Email messages and alerts	Special requirements in all use case specs	BR09
CCC24	User security levels	Special requirements in all use case specs	BR09
CCC25	Employees data encryption for secrecy	Special requirements in all use case specs	BR09
CCC26	Making transaction, loan/ advances etc	Special requirements in all use case specs	BR09
CCC27	Errors intimation	Special requirements in all use case specs	BR09
CCC28	Transaction logs	Special requirements in all use case specs	BR09
CCC29	Calendar planning	Special requirements in all	BR09

		use case specs	

Table # 2

5.1 Discussion

CCC02 is identified by applying rule-BR05 (Inclusion use case). This inclusion which made by use cases "Hire employees" and "Employees Left" depicts that use case "Calculate wages/ salary" crosscuts other two concerns which thus indicates crosscutting concern. Both the uses cases "Hire employees" and "Employees Left" need this inclusion to achieve each goal. But the concern presented by "Calculate wages/ salary" is crosscutting and its accuracy is greater as a company's concern instead of an employees' concern when an employee decided to join the company and left the company for some reason. Such evaluation must be checked for correctness to assess the profitability of the company which is in both cases not the goal of primary actor of the system under design.

Same is the case with CCC01, which is also coupling; where rule-BR04 "Extension use case" is applicable because this is a sub-goal of use case "Hire employees" which assures that future hiring is possible so it must be focused although it is not possible at the current stage. As guarantee concept given by Cockburn [7], there must be minimum guarantee for every use case which is at base and every distinct behavior or significant guarantee may be separated by a use case instead of going into details to set as an alternative flow.

Rule-BR03 "Alternative flow" is applied to both CCC03 and CCC06 and they both are identified as crosscutting concerns. This alternative flow is activated if there is lack of information in each case and rule is defined in business context. The scenario of CCC03 is that no candidate can apply against an already filled post or a candidate cannot apply for the same post twice. This flow has a hidden business rule thus this is a crosscutting concern requirement because this will stop candidates to append for a job without a sort of verification.

Crosscutting concerns CCC04, CCC07, CCC08, CCC10, CCC11 & CCC17 are related to the requirements where candidate withdrawal from the required post or his/ her appointment/ hiring is need to be cancelled. Such requirements are essential part of any system where users are going to interact with the system and should be focused attentively as flaws in such requirements may lead towards an unsuccessful software product. As these requirements can act as a primary actor concern but owing to the reason that these types of requirements opposes the main goal of the system so should be taken as crosscutting concerns.

Rules -BR03 "Alternative flow" is also applied to CCCC05, CCC09, CCC14, CCC15 & CCC16 and these concerns are identified as crosscutting concerns. Alternative flows are such flows of system where there is a need to study the system under design in more detail or where there information is not available completely but keep in mind that all the alternative flows ultimately will meet the basic flow somewhere in the system under design and these alternative flows are included for exceptional cases only. Because alternative flow may present anywhere in the system under design, so it will be considered as crosscutting concerns. Alternative flows are sub-goals of the main goal crosscut it in special circumstances to achieve the targets of main goals. Just to give an example of the system under study, CCC09 is a such type of flow which is focused on the best interest of the candidates and if there is a vacant post in the company, this will automatically intimate the candidate that now he/ she can walk for an interview vice versa. This flow also takes responsibility to send and intimation to the first candidate in waiting list.

Rule-BR08 "Business rule" is applied to CCC12, CCC13, CCC19, CCC20 & CCC21 and these concerns are identified as crosscutting concerns. For example, CCC20 & CCC21 are designed to presents such rules which are in the best interest of the employee so he/ she may remain an active member of the company as well as a tangible asset of the company. Such rules help to progress the company in a sense that the employees make the good repute of the company and they want to work for the company for years and years. Use cases "Calculate wages/ salary", "Employees Left" & "Manage employees" are under such rules for classification purpose or to give them incentives like bonus or increments. Although, these rules provide certain benefits to the employees and problem for the company from finance view point but also work as a stimulator for the employees to increase their performance, loyal to the company and to enhance good repute of the company as well. Thus these concerns can be taken as crosscutting concerns as they crosscut the main goal of each of the use case.

Rule-BR09 "Non-functional requirements" is applied to CCC23, CCC24, CCC25, CCC26, CCC27, CCC28 & CCC29 and these concerns are identified as crosscutting concerns. All use cases are using these non-functional requirements. Example given: CCC25 is crosscuts as it addresses the all operation related to the data communication.

Rule-BR07 "Post-condition" is applied to CCC22 and this concern is identified as crosscutting concern as it has

same goal in two use cases "Hire employee" and "Employee left". Such post-conditions are added to send a notification at relevant times to the relevant persons to intimate him/ her for whether he/ she is "hired" for a job or "left" the job so that such condition must be satisfied. In a sense, this concern has little bit similarities with CCC23 & CCC09 but it arises from different cause. CCC23 is about sending intimation to the candidate/ employee via email which is a technology based factor but the intimation/ notification can be sent via other sources but in this case mean of sending intimation/ notification is purely based on technologies. CCC09 is about handling candidates whether they are waiting for a vacancy, waiting for the wages/ salary collection or at the time of "left" from the company. In this scenario, candidate is waiting for to get a chance for vacant post and at the same time speedy availability of a suitable candidate for the vacant post from company's perspective which actually crosscuts it for the satisfaction of a specific concern and not linked with the output which is actually sending intimation/ notification. This type of requirements can be seen through CCC22 which is represented to send intimation/ notification at a certain time in the system under design.

From the discussion it can be said that there are some aspects which crosscut other aspects so the aspects being used to send email by encapsulation should relate to the aspect that are used to send notification by encapsulation. Same is the case with employees handling concern in "Employee left" and thus bridging to perform certain behaviors of the system under design.

5.2 Implementation

A Software house was requested to implement the software under design in JAVA language while software specs were kept under study. The plan was to code each identified crosscutting concern as a separate aspect. Specific purpose AspectJ [13] - figure # 3 and Hibernate [14] and Swing-Bean framework [15] - figure # 4 was used for object-relation/ mapping, database access and user interfaces building.

Although it was almost a full scale project but many simplification were made that were unrelated or would not bring different results, to keep the research within time schedule and budget. Simplification like only one type of user to control "Hire employee", "Employee left" and Employees' "IN" and 'OUT" and instead of web based user interface, a simple desktop interface was developed.

As CCC01& CCC06 were brought together in single code as former is an extension to the later one. CCC04 & CCC07 were excluded from the implementation due to only a minor output of "no" option. CCC05 & CCC25 were related to the web implementation and as web implementation was previously simplified to desktop interface that's why both of these concerns were eliminated from the implementation. Hibernate use eliminated the implementation of CCC26 as Hibernate has its own built in transaction control functions. Crosscutting concern CCC28 was also eliminated from the implementation because log has nothing to do with the results of this research. Crosscutting concern CCC22 is purely based on the nature and demand of technology which would be in use for sending intimation/ notifications thus has nothing to do with the results of this research that's why also excluded from the implementation. CCC11 & CCC17 were used in the implementation to show the better approach. As we can see, they both present cancellation or delete functions and a minor change was made in their requirement: a confirmation dialog box was added at the time of deletion to confirm whether the user wants to continue with the delete option or not. Only the vital crosscutting concerns were implemented as different aspects for the purpose to evaluate the technique more accurately, efficiently.

5.3 Evaluation

Every technique should be tested after implementation to see whether the results are according to the expectations. Evaluation and testing [20] is also necessary to determine whether all the crosscutting concerns that were identified were really the crosscutting concerns and actually modularized or not. Many of such questions were evaluated through ConcernMorph [16] - figure # 2. ConcernMorph is metric-based software which detects crosscutting concerns based on ConcernMapper. ConcernMapper is a simple Plug-in for Eclipse IDE that facilitates to map between classes, methods & their related fields. Mapping which was created by ConcernMorph is based on implementation concern and ConcernMorph can evaluate many metrics and can identify crosscutting concern. It can also deal to identify and name the crosscutting concerns as one of famous crosscutting concern types.

The first step of evaluation was to identify concern and then their mapping with classes, methods and fields. As we have simplified some details and irrelevant requirements thus it was easy to pick concerns because every use case presents some functions of the software under design and thus use cases can be used as concerns. Identified crosscutting concerns are natural candidates. Although non-functional requirements can also be used as

concerns but we have already taken them as crosscutting concerns. 25 concerns were picked and every concern had his implementation map. While using ConcernMorph, only 3 out of 25 were identified as crosscutting concerns and these are use cases of "Hire employee", "Employee left" and "Calculate wages/ salary".

An implementation issue was found during "Calculate wages/ salary" use case: it was already been detected as crosscutting concern and owing to this reason it was implemented as a concern. The departments' domain class carries department daily rates and the code was scattered by ConcernMorph.

For another two use cases, a same situation was faced. There were many input screens to get input data from user for example employees' daily "IN" and "OUT", employees' identification at different stages during his/ her stay at the company and by implementation such concern as distinct classes and the said tool detected a scattered code here as well.

The results of evaluation were very good as they showed that all the implemented crosscutting concerns were modularized and this that after all simplifications and exclusion, only four use cases were crosscutting free namely "Manage employees", "Manage departments", "Hire employee" and "Handling waiting list". Other three use cases crosscutting have implementation related specific issues. Furthermore, an improved description for simplification of CCC11 and CCC17 should be done.

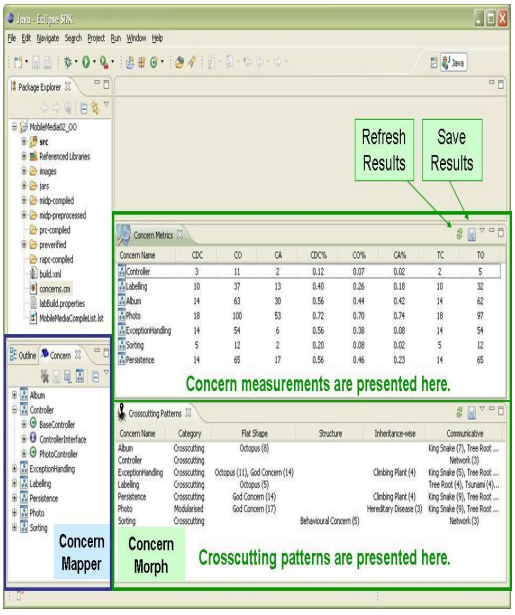

Figure # 2

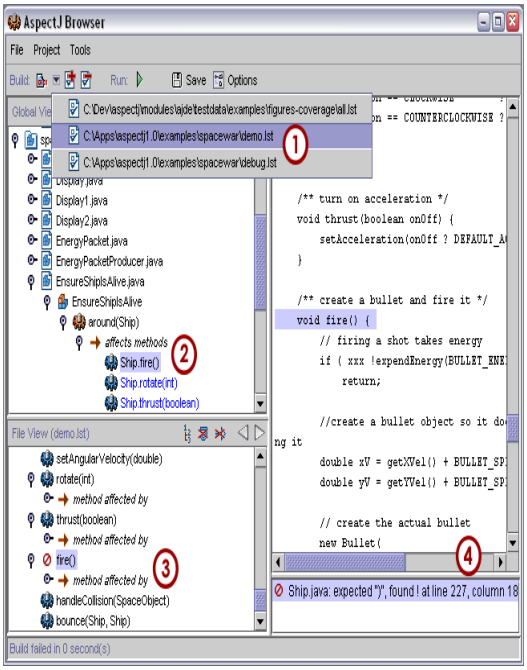

Figure # 3

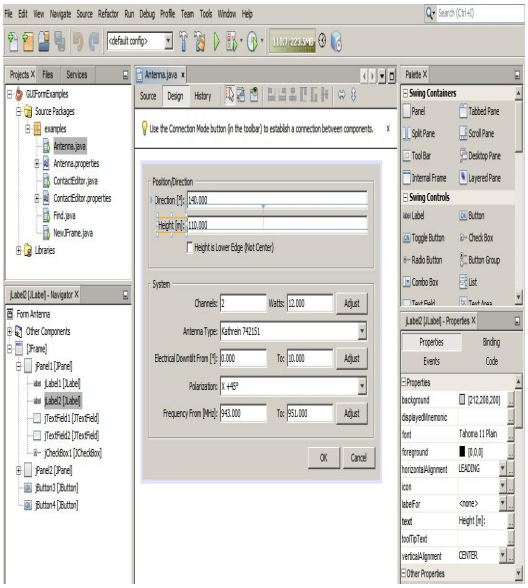

Figure # 4

6 Related work

Early identification of aspects in Aspect-Oriented Software Development is not entirely focused here in light of use cases. This research focused on identification and modeling [19] of aspects according to uses cases life cycle [8]. It recommends slice and use case modules and new entities of use case models. Relation between different use cases can be modeled through these entities, which further

gives inter-type declaration and pointcuts modeling. As this is not focusing on early identification of aspects that's why might not present bright approach for aspects identification.

Sousa et al. [17] presents a method that is closely non-functional requirements oriented by adapting NFR Framework [18] for identification and mapping of crosscutting concerns. It also purposes a relationship of use cases namely "crosscut" which would be helpful in extension use cases, where extender is not depends on the main goal of the use case and also use case is not specific to the base case. This paper works on the usual relationship of use cases owing to the reason that it would help to understand the system under design and same is the aim of eliciting of requirements process. As crosscutting concerns are indicated by extension use cases thus motivation for such relationship shows that there are additional features presented by extender that are not important for main goal that's why these uses case are not included in the scenario.

Till now, no method deal with use case specification specific requirements but there is only a presentation of relationship of different use cases and their structures. Additional information regarding identification of crosscutting concern through use cases is not possible as use cases are only atomic units of the requirement specifications.

7 Conclusion

Our work has tried to present a case study based approach to better identify crosscutting concerns. In our research, we have tried to include different aspect and almost every property of use cases to get sufficient knowledge about crosscutting concerns. Our approach did not plan to find software solution that may apply according to these identifications and not the capability to find all the crosscutting concerns as well.

On the other hand, by using this approach, most of crosscutting concerns can be identified and this approach can be very useful for an organization/ company where use cases are used as requirement engineering technique. Our main goal was to propose a knowledgeable way to support in crosscutting concerns' identification in early stages of software life cycle for better requirement specification gathering and further refinements.

By evaluating this approach good results were collected after implementation and the gathered metrics point out towards the fact that the total number of assumed crosscutting concerns has reduced by implementation through this approach.

This approach can work with any existing software system which has used case based requirement engineering processes [8]. This approach can also help the newbie's to better predict and identify crosscutting concerns in a better way as this is the easiest approach ever used and thus it's very easy to work with this approach with limited resources and without introducing any sort of new software process. Adaptation of this approach is very easy but the outcome is fruitful.

Future work would include further phases of software development life cycles i.e. parameters and methods for analysis and design, Unified Markup Language (UML) analysis, classifying crosscutting concerns according their importance and impact factor in the software system under design for better assumption for their inclusion or exclusion and metaphors for better understanding crosscutting concerns and their impact on the system under design and effects on Software Life Cycle (SLC).

References

[1]. G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. pages 220–242. 1997.

[2]. A. Przybyłek. Where the Truth Lies: AOP and Its Impact on Software Modularity. FASE 2011, LNCS 6603, Pages 447-461, 2011.

[3]. B. T. e. M. A. Jethro Bakker. Characterization of early aspects approaches. In Proceedings of the Early Aspects Workshop at AOSD, Netherlands, 2005.

[4]. A. Rashid. Aspect Oriented Requirements Engineering: An Introduction. pages 306-309. IEEE, 2008.

[5]. S. Busyairah Ali and Z. Kasirun. An approach for crosscutting concern identification at requirements level using NLP. International Journal of the Physical Sciences Vol. 6(11), pages 2718-2730, 2011.

[6]. E. Baniassad and S. Clarke. Theme: an approach for aspect-oriented analysis and design. Pages 158–167, 2004.

[7]. A. Cockburn. Writing Effective Use Cases. Addison-Wesley Professional, January 2000.

[8]. I. Jacobson and P.-W. Ng. Aspect-Oriented Software Development with Use Cases (Addison-Wesley Object Technology Series). Addison-Wesley Professional, 2004.

[9]. P. Metz, J. O'Brien, and W. Weber. Specifying use case interaction: Types of alternative courses. Journal of Object Technology, 2(2):111–131, 2003.

[10]. K. E. Wiegers. Software Requirements. Microsoft Press, Redmond, WA, USA, 2003.
[11]. A. M. d. C. Antonio Maria P. de Resende, Fabio Fagundes Silveira and H. A. X. Costa. Meaid: A method for early aspect identification and definition. 2008.
[12]. M. A. Cibran, M. D'Hondt, and V. Jonckers. Aspect-oriented programming for connecting business rules. Pages 306–315, 2003.
[13]. I. Kiselev. Aspect Oriented Programming with AspectJ. 2002.
[14]. J. Elliott. Hibernate: A Developer's Notebook. 2004.
[15]. R. Johnson. Introduction to the Spring Framework. 2005.
[16]. A. G. Eduardo Figueiredo, Jon Whittle. Concernmorph: Metrics-based detection of crosscutting patterns.
[17]. G. Sousa, S. Soares, P. Borba, and J. Castro. Separation of crosscutting concerns from requirements to design: Adapting the use case driven approach. In In Proc. Early Aspects Workshop at AOSD, 2004.
[18]. L. Chung1 and J. Cesar. On Non-Functional Requirements in Software Engineering. Mylopoulos Festschrift, LNCS 5600, pages 363–379, 2009.
[19]. J. Evermann. A meta-level specification and profile for aspectj in UML. In AOM '07: Proceedings of the 10th international workshop on Aspect-oriented modeling, pages 21–27, New York, NY, USA, 2007.
[20]. A. Rashid, T. Cottenier, P. Greenwood and R. Chitchyan. Aspect Oriented Software Development in Practice, Tales from AOSD, Europe. IEEE Computer Society, 2010.