Aspect-Oriented Requirements Engineering for Advanced Separation of Concerns: A Review

Narender Singh and Nasib Singh Gill

Department of Computer Science and Applications
Maharshi Dayanand University, Rohtak - 124001, Haryana, India.

Abstract

Software engineering was introduced to cope with software crisis with two fundamental principles: separation of concerns and modularity. Many programming paradigms have been proposed and used while considering the fundamental principles from the early days. Complex software systems were successfully modularized but complete separation of concerns is still impossible to achieve using today's most popular programming paradigms such as object-oriented programming. There are some concerns which could not be separated properly in a single class or module due to their highly coupling with other classes or modules' behaviors. We call such unmodularized concerns as crosscutting concerns and these are responsible for scattering and tangling.

Aspects are the natural evolution of the object-oriented paradigm. They provide a solution to some difficulties encountered with object-oriented programming, sometimes scattering and tangling. Hence, Aspect-Oriented Software Development (AOSD) is another step towards achieving improved modularity during software development. It gives emphasis to the separation of crosscutting concerns i.e. advanced separation of concerns and encapsulation of crosscutting concerns in separate modules, known as aspects. It later uses composition mechanism to weave them with other core modules at loading time, compilation time, or run-time. Aspect-Oriented Requirements Engineering (AORE) is an early phase in AOSD that supports separation of crosscutting concerns at requirements level. It does not replace but rather complements any of the existing requirements methodologies.

Over the last few years, several research efforts have been devoted to this area. Several techniques for crosscutting concern identification have already been proposed. In this paper, an attempt is made to review the existing approaches and understand their contribution to requirements engineering.

Keywords: *Separation of Concerns, Crosscutting Concerns, Aspect-Oriented Software Development, Aspect-Oriented Requirements Engineering.*

1. Introduction

The term "*Software Engineering*" was introduced in the NATO Working conference [1] on Software Engineering in 1968 to cope with *software crisis*. A number of approaches have been proposed to deal with the software crisis. Developing the complex software systems became easy. However, the progress in software engineering concepts did not keep track with increasing complexity of modern software systems. It is difficult to meet current and future needs in software development using today's most popular programming paradigm such as object-oriented programming.

According to the Standish Group in 1995 [2], only about 16% of software projects were successful, 53% were full with problems (cost or budget overruns, content deficiencies), and 31% were cancelled. The Standish Group's just-released report, "*CHAOS Summary 2009* [3]", only 32% of all projects succeeding which are delivered on time, on budget, with required features and functions" says Jim Johnson, chairman of The Standish Group, "44% were challenged which are late, over budget, and/or with less than the required features and functions and 24% failed which are cancelled prior to completion or delivered and never used". Evidence suggests that despite the improvement from 1995 to 2009 the current situation in software development is far from adequate.

Separation of concerns and *modularity* are the fundamental principles that drive the research in software engineering since the early days. The term "*separation of concerns*" was introduced by Edsger Dijkstra, to refer the ability of identifying, encapsulating and manipulating parts of software that are crucial to a particular goal or purpose in his book "*A Discipline of Programming*" [4]. The basic idea behind separation of concerns is to handle one property of a system at a time. In other words, a complex problem that is hard to understand should be divided into a series of smaller problems; those are less complex and easier to handle by the designer. These smaller problems may then be designed one at a time by different designers and finally integrated to solve the big problem. *Modularity* [5] [6] is the principle to structure software into modules where modules are self-contained, cohesive building blocks of software. A module is a device to implement a concern and modularity is a consequence of separation of concerns.

Many programming paradigm have been proposed with keeping the fundamental principles in mind. It becomes possible to modularize the complex software systems. But, it is still difficult to achieve complete separation of concerns using today's most popular programming paradigm such as object-oriented programming. There are some pieces of processing i.e. concerns that did not seem to fit in any particular single classes. This is because they are too tightly coupled to the behaviors in many other classes or modules. These unmodularized concerns are called as crosscutting concerns and are responsible for scattering: the implementation of a concern is spread over several program modules and tangling: a program module implements multiple concerns. Several empirical studies provide evidence that crosscutting concerns degrade code quality because they negatively impact internal quality metrics such as program size, coupling, and separation of concerns [7].

Aspects are the natural evolution of the object-oriented paradigm. They provide a solution to some difficulties encountered with object-oriented programming, sometimes scattering and tangling. AOSD [8] is another step towards achieving improved modularity during software development. It focuses on *crosscutting concerns* by providing means for their systematic identification, separation, representation and composition [9]. It encapsulates crosscutting concerns in separate modules, known as *aspects*. It later uses composition mechanism to weave them with other core modules at loading time, compilation time, or run-time [10].

AOSD was introduced first at programming level, with Aspect-Oriented Programming, where aspects are handled in code. A number of Aspect-Oriented Programming approaches have been proposed. Work has also been carried out incorporate aspects, and hence separation of crosscutting concerns, at the design level mainly through extensions to the UML metamodel [11] [12] [13]. However, crosscutting concerns are often present before the solution domain, such as in Requirements Engineering [14] [15] [16].

AORE [17] is still an emerging field with many open research issues. Research in the early phases of software development with aspect-oriented paradigm has been increasing. Handling crosscutting concerns in the early stages of software development is beneficial rather than handling them in later stages of software development because it not only makes the design simpler, but also helps to reduce the cost and defects that occur in the later stages of development. AORE focuses on identifying, analyzing, specifying, verifying, and managing the crosscutting concerns at the early stages of software development. It does not replace but rather complements any of the existing requirements methodologies.

In this section, an attempt is made to highlight the concept of AORE for advanced separation of concerns i.e. AORE for handling crosscutting concerns at the early stage of software development. Section 2 reviews the existing literature by many researchers. A roadmap to research is discussed in section 3. Finally, we conclude in section 4.

2. Literature Review

The success of a software system depends on how well it fits the needs of its users and its environment [18]. Software requirements comprise these needs, and *requirements engineering (RE)* is the process by which the requirements are determined [19]. Successful RE involves understanding the needs of users, customers, and other stakeholders; understanding the contexts in which the to-be-developed software will be used; modelling, analyzing, negotiating, and documenting the stakeholders' requirements; validating that the documented requirements match the negotiated requirements; and managing requirements evolution [20]. Existing requirements engineering approaches, such as use cases [21], viewpoints [22], and goals [23] provide good support for identification and treatment of some kinds of requirements. However, these requirements approaches do not explicitly support well broadly-scoped requirements, such as crosscutting concerns, and do not explicitly support their composition. Moreover, they all suffer from the "*tyranny of the dominant decomposition*" [24]. AORE, therefore, complements these approaches by providing systematic means for handling such crosscutting concerns.

Over the last few years, several research efforts have been devoted to developing AORE models that can help in extracting, identifying, and modeling aspects in the early phase of requirements analysis. In the following we discuss briefly these efforts.

Grundy's [25] proposed an approach called Aspect-Oriented Component Requirements Engineering (AOCRE) that focuses on identifying and specifying the functional and non-functional requirements relating to key aspects of a system each component provides or requires. In component-based systems, applications are built from discrete, inter-related software components, often dynamically plugged into running applications and reconfigured by end users or other components. Some components may have many aspects and others a few. Aspects may be decomposed into aspect details. Candidate components are found from OOA diagrams, by reverse engineering software components, or bottom-up consideration of individual, reusable components. The AOCRE process begins with identifying components' aspects, where for each component, we identify aspects for which the component provides services or requires services from other components using possible stakeholder requirements and object services. After identifying a component's aspects, we can reason about components and aspects. Further, we can infer inter-component relationships that allow engineers to reason about the validity relationships and aspects specified. Aspect-oriented component requirements also assist components design and implementation. They provide a focused set of functional and non-functional constraints for refining design, and provide a specification that an implementation can be tested against. AOCRE exhibit improved reusability and extensibility, and systems built with these components exhibit improved allocation of responsibility for data and behaviour among both reused and application-specific components. This approach is too specific for component development, not showing evidence of its use in software development in general. Besides, the identification of aspects for each component is not clearly defined and lacks tool support.

Rashid et al. [26] proposed a model for aspect-oriented requirements engineering. The model supports separation of crosscutting properties from early stages of the development and identification of their mapping and influence on later development stages. Now, it is possible to identify conflicts, establish possible tradeoffs, and promotes traceability throughout the system development and its evolution. The early separation of crosscutting concerns improves modularisation and hence, it is possible to build flexible and adaptable systems that meet the needs of volatile domains such as banking, telecommunications and e-commerce. The model supports separation of crosscutting properties from early stages of the development and identification of their mapping and influence on later development stages. But it lacks on validation of aspects, their composition with other requirements and resolution of possible conflicts resulting from the composition process. It also lacks a notation to describe aspects, their interactions and composition relationships at the requirements level.

Baniassad and Clarke [27] proposed the theme approach that provides support for aspect-oriented software development at two different stages. Theme/Doc, which is used for viewing and analyzing the requirements at requirements phase; and Theme/UML, which allows a developer to model features and aspects of a system, and specifies how they should be combined at design phase. In the theme approach, a theme is an element of design, which is a collection of structures and behaviours that represent one feature. Multiple themes can be combined or integrated to form a system. Themes are further classified as base themes, which may share some structure and behaviour with other base themes, and crosscutting themes which have behaviour that overlay the functionality of the base themes. Crosscutting themes are known as aspects. Action view is used to identify crosscutting behaviours. To create an action view, two inputs are needed: a list of key actions identified by the developer by looking at the requirements document and picking out sensible verbs, and the requirements as written in the original document. Theme/Doc then performs lexical analysis of the text and generates the action view. The theme approach involves three main activities as finding themes, modeling and composing themes, and checking Themes/UML. The process begins with finding themes. Here, we identify actions and generate an action view to examine their relationships. After analysis of the view, we determine that all of these actions will not be modelled as separate themes. Instead, we determine the relationships between the actions to decide how to group the actions into larger themes. Here, we also determine which themes are crosscutting and which are base. The next step in theme process is modelling and composing themes. Here, the theme view is used to drive the modelling and composition semantics for design using Theme/UML. To ensure that the developer carefully considers the order in which crosscutting themes are composed with base themes, Theme/UML allows only one crosscutting theme per composition. We therefore needed to inspect the crosscutting relationships to determine the order of binding. For this we used the clipped action view. In this view, the themes are positioned hierarchically, based on whether they crosscut one another. Finally, we check the validity of the design choices we made. This approach supports effective aspect identification, requirements coverage, traceability, and scalability of action views. However, this approach is only applicable for structured requirements document. As for the developers, they must possess the domain knowledge. Hence they must go through the whole requirements source document to identify the crosscutting concerns. They have to manually map the relationship between the themes and requirements. It is costly and time consuming to handle large amount of requirement sources.

Whittle et al. [28] proposed an approach to model scenario-based requirements using aspect-oriented paradigm. The main focus was on representing aspects during use case modelling. The approach provides a way to describe aspectual and non-aspectual scenarios independently and then merge them together to validate the complete set of scenarios. Aspectual scenarios, i.e., scenarios that crosscut other scenarios, are modeled as interaction pattern specifications (IPSs) and non-aspectual scenarios are modeled as UML sequence diagrams. Each aspectual and non-aspectual scenario is then translated into a set of state machines. The next stage of the process composes the aspectual and non-aspectual state machine for each entity. The result is an executable set of state machines that completely describe the requirements and in which aspectual and non-aspectual behaviour has been merged. Validation of these state machines can now take place using either a simulation harness or a code generator and the results can be feedback into the overall process. Composing (or weaving) aspectual and non-aspectual state machines helps the requirements engineer grasp the full picture. The approach supports better modularization and traceability but it lacks to address scalability. The developer must provide binding statements for each aspect and for each scenario that the aspect crosscuts. It also not proposed any systematic technique to aspectual scenarios identification.

Jacobson et al. [29] proposed an approach called aspect-oriented software development with use cases to handle crosscutting concerns. This approach is an extension to the traditional Use Case approach proposed by the same author and introduced new concepts like use case slices, extension use case, and pointcut. Here, a system is built use case by use case. The process begins with identifying use cases. Further, we need to specify each use case, to analyze it, and to design use cases in terms of use case slices and use case modules. Use case slices are used to employ aspects. Use case modules are used to contain the specifics of a use case over all system models. Extension use cases are the special kind of use cases that contain additional functionality of the use case. We implement and compose them using a composition mechanism to weave them at loading time, compilation time, or run-time to form a complete system. The approach includes processes like identifying, specifying, analyzing, designing, and implementing use cases. The approach strongly related to UML; but lacks in conflicts handling.

Moreira et al. [30] proposed an approach called concern-oriented requirements engineering (CORE), which treats each concerns uniformly. Here, a concern is any coherent collection of requirements. They also not classified concerns into viewpoints, use cases or aspects though their concerns encapsulate the coherent sets of functional and non-functional requirements. Concern space at the requirements level is represented as a hypercube. Each face of the hypercube represents a particular concern of interest. The process begins with identifying and specifying concerns using existing requirements elicitation mechanisms such as such as viewpoints, use cases, and goals. The identified concerns are specified using well-defined templates. The second step is to identify coarse-grained relationships among concerns by relating concerns to each other through a matrix. These relationships are identified using techniques such as domain analysis, ethnography, and natural language processing. The third step is to specify the possible projections of each concern on other concerns, which is achieved through composition rules. The fourth step is to identify and resolve conflicts (if any) among the concerns. This is achieved by building a contribution matrix, where each concern may contribute positively or negatively to the others. Prioritisation mechanism is used to solve conflicts and helping negotiation and decision-making. The last activity is to identify the dimensions of concerns. There are two dimensions of a concern at requirements level, which are mapping and influence. This approach supports multi-dimensional separation of concerns, which treats each concerns uniformly, hence, avoiding the dominant decomposing. It also establishes early trade-offs and solve conflicts that help negotiation and decision-making. But, it does not focus on the exact kind of relationships between two concerns, validation of proposed model with more case studies, and setting the concern specific actions and operators.

Araujo et al. [31] proposed an approach that incorporates aspect-oriented concepts into agile software development at requirements level. Agile software development aims at fast communication and incremental delivering of software artefacts. The aspect-oriented agile requirements approach focuses on defining and modelling initial crosscutting requirements as scenarios. Scenarios are descriptions of desired or existing system behaviour. Scenarios are commonly used in requirements engineering because they are easily understood by all stakeholders. Scenarios may crosscut other scenarios. Crosscutting scenarios are called aspectual scenarios. First, we begin with identifying main functionalities and refine those using scenarios. Secondly, we need to identify aspectual scenarios. This is achieved by analyzing all scenarios and observing some behaviour that crosscut several scenarios. Third, we need to compose aspectual and non-aspectual scenario definitions. Compositions are specified through simple rules. Finally, by analyzing the composition rules we may find ambiguities, errors omissions and conflicts in our scenarios. This analysis can be realized through inspections. Also, the participation of the stakeholders is crucial. Conflict identification can be accomplished by adapting the existing mechanism. The final set of aspectual and non aspectual scenarios plus composition rules are used to implement users' functionalities. The approach lacks on complete composition mechanism, conflict identification mechanism, validation, and tool support to guarantee that the approach will be used in an agile fashion.

Brito et al. [32] proposed an integrated approach for aspectual requirements to handle separation, modularization, representation and composition of concerns. The approach defines three main tasks, each one divided in several subtasks. The first task is identifying concerns, which aims at identifying the concerns of a system. A concern can be defined as a set of coherent requirements that the future system must have. This can be accomplished by analyzing the initial requirements, transcripts of stakeholders' interviews, etc. The second task is specifying concerns, which consists of many subtasks such as collecting information about concerns, specifying them using a template, and to design models like UML use case, interaction and class diagrams. The final task is composing concerns incrementally until the whole system is obtained. Each composition takes place in a match point in the form of a composition rule. A match point tells us which concerns (crosscutting or non-crosscutting) should be composed together. A composition rule shows how a set of concerns can be weaved together by means of some pre-defined operators. In order to accomplish this, we need to identify crosscutting concerns. The approach defined the main concepts as an extension of the UML metamodel, which allows a developer to better capture, analyze and understand the approach. The tool facilitates the specification of concerns, identification of crosscutting concerns, generation of the match point table and definition of composition rules. The approach does not define any method and tool with a reference model to support forward and backward traceability.

Z. Jingjun et al. [33] proposed aspect-oriented requirements modelling aiming to apply AOP paradigm at requirements engineering stages of software development. This approach supports separation of concerns both functional and non-functional, and modelling them in UML with class diagrams and state-chart diagrams respectively. The process includes five activities as follow: identifying and specifying concerns, analyzing concerns, composing concerns, weaving concerns, and simulating and validating requirements. First, identify both functional and non-functional concerns from system requirements, and then specify them in UML as OOP class and aspect class. Second, analyze the relation among concerns by detecting and removing the conflicts if any. Third, compose concerns by describing the static structure of the system. Next, during weaving concerns, the whole state-chart diagram of the system is given, and then finishes the weave process. Finally, simulate the system function with the whole state chart, and validate the function whether it meets the system requirements or not. If not, return to the first activity, identify and specify concerns again. Or, complete the model process. This model supports separation and modelling of concerns. It also supports an effective method to solve the mismatch among the aspects, which reduces the complexity of the system and increases software's reusability and maintainability. It uses terms functional and non-functional concerns as core and crosscutting concerns respectively. But, a crosscutting concern may be functional as well as non-functional. So, this method does not clearly identify and specify crosscutting concerns which are functional.

Chitchyan et al. [34] proposed Requirements Description Language (RDL), which is a symmetric AORE approach. It modularizes the requirements in symmetric fashion and represents them using the same abstraction, i.e., a Concern, to represent both crosscutting and non-crosscutting elements. A concern may be simple i.e. containing only requirements or composite i.e. containing requirements and other concerns. Both concern and requirement can be described as multi-sentence elements; where an element can be a subject or an object or relationship. A subject is described as an entity that undertakes actions and in RDL; it corresponds to the grammatical subject in the clause. An object is described as an entity that is affected by the actions undertaken by the subject of the sentence. In RDL, it corresponds to the grammatical object in the clause. A relationship is described as the action performed by the subject on or with regards to its object(s) and can be expressed by any the verbs or verb phrases in natural language. The main semantic load is carried by subject-relationship-object structure. The subject and object denote the entities of significance in it, whereas, the interactions between these entities are reflected by relationships. In this approach the relationship denotes the most central function, as it defines the functionality and/or properties that the subjects and objects provide. When specifying requirements stakeholders often qualify how important or significant a specific functionality or property is to them. In the RDL such qualifications are represented by the Degree element. Degree element depicts the strength of the relationship between the subject and object. The RDL elements discussed above are used for requirements description. The next activity is element composition. The assembling of separately defined requirements modules aiming to ensure their desired interactions or addressing undesired ones is termed as composition. Here, three sub-elements of a composition element are constraint, base, and outcome. A Constraint element specifies checks and restrictions applied on a set of requirements, and the action taken in imposing these constraints. Base element provides a query for selecting the set of requirements that are affected by some constraints; and the temporal or conditional dependency between these requirements and the constraints. The Outcome element defines how to treat the imposition of constraints upon the base sets of requirements. Composition specifications are written based on these semantics rather than requirements syntax, hence providing improved means for expressing the intentionality of the composition, in turn facilitating semantics-based reasoning about aspect influences and trade-offs. However, the approach requires a complete and precise requirements document, which can't be expected before requirements elicitation and analysis. Also, to validate this approach, it is still required to refine requirements, identify and resolve conflicts and trade-offs.

Jing Zhang et al. [35] proposed an aspect-oriented approach to supporting separation of crosscutting concerns in activity modelling. Aspect-specific constructs have been introduced as an extension to the activity models. Activity modeling describes the behavioural aspects of a system and is used to define a computational process as control flow and data flow among its constituent actions. It consists of many kinds of nodes and edges. The sequencing of actions is controlled by control flow and object flow edges. An activity node can be an action, an object node or a control node. An object node holds data that flow through the activity model. Control nodes are responsible for routing control and data flows in an activity. Activities can be divided into different partitions that represent different kinds of activity groups for identifying actions that have some characteristics in common. Activity specifications grow with increasing complexity of the system and require lifecycle maintenance for the concerns that crosscut different activity modules. Aspect-oriented paradigm provides a solution to above problem by encapsulating crosscutting concerns in specialized units called aspects. New concepts like joinpoints, pointcut, and advice are introduced here. A joinpoint specifies "where" the crosscutting concern emerges in the activity model. A pointcut is defined as a special construct containing a group of particular join points, which defines a pattern to identify matching join points. In activity modeling, the concern behaviour is implemented using an activity model referenced by a special action called advice, which specifies "what" (i.e., the behaviour) makes up the crosscutting concern. This paper applies an aspect-oriented approach to supporting separation of crosscutting concerns in activity modeling. Aspect-specific constructs have been introduced as an extension to the activity models. The current implementation of the pointcut specification only allows join point to be referred to action nodes. The approach lacks on covering other kinds of activity nodes and investigating more advanced pointcut selection patterns.

Budwell and Mitropoulos [36] proposed a methodology called Structured Lexicon for Aspectual Identification (SLAI) that is based upon the Language Extended Lexicon (LEL) for capturing requirements. LEL is used for vocabulary acquisition or understanding problem language. LEL consists of three elements: signs, notions, and behavioural responses. Signs represent any word or phrase that has a special meaning within the Universe of Discourse (UofD). Notions and behavioural responses help to define the meaning of each sign. Notions define the signs in the context of the UofD, whereas behavioural responses define how the sign is used. The principle of circularity and minimal vocabulary is used to define signs. The process begins with clearly identifying and defining aspects at the requirements phase of software development. An aspect is a crosscutting concern or matter of interest in a software system. A concern must be derived from either functional or non-functional requirements. The next step is to capture crosscutting requirements (functional or non-functional) as aspects. Functional requirements can be identified as aspects in the form of UML use cases. A functional aspect is defined as a crosscuts use case. This is a new type of use case in UML after modifying extends and includes use cases. Non-functional requirements are converted into operationalizations and then into use cases. This methodology is divided into two flows, functional requirements flow and non-functional requirements flow. In functional requirements flow, actors of the system are identified first. Next, use cases are identified and detailed them into use case steps. Later, the use case steps are recorded in the SLAI. In non-functional requirements flow, SIG graph is used where the roots of the SIG graph represent non-functional requirements and leaf nodes for each path represent operationalizations, which are either operations or design constraints. Since design constraints tend to be considerations that need to be addressed in the design phase, they are not included in the SLAI. The use case information is not enough to classify crosscuts use cases as aspects. Thus, for each crosscuts use case, a table is generated that lists each use case with which it interfaces. For each of these use cases listed, the condition of the extension, the composition rule operator, and the affected point are defined. The approach supports the identification of aspectual or crosscutting use cases from both functional and non-functional requirements as well as systematically identifying both aspects and potential aspects by the use of a limited set of vocabulary for the terms defining the requirements of the system. From the study, it is observed that the methodology lacks on aspect composition as well as conflict resolution.

Ali and Kashirun [37] developed a model to identify crosscutting concern and designed an automated tool based on the model. The model is based on the proposed approach which is adapted from Theme/Doc and Early Aspects Identification approaches. The model is fully automated and involves non-collaborative processes unlike conventional requirements engineering processes, which are generally collaborative and iterative in nature. The execution of processes in this model is sequential where each process requires output of the previous process as input for execution. The model consists of processes like structuring requirements, removing redundancy, part-ofspeech (pos) analysis, semantic analysis, filter verbs identification, map relationship view, refining the relationship view, identifying dominating verb, and modelling crosscutting influences. Structuring requirements task involves numbering all the requirements agreed by the stakeholders to identify and manipulate each requirement uniquely in the next stages. Sometimes, same requirements are specified many times by different stakeholders. The redundant requirements are eliminated during removing redundancy process. Verbs are extracted from each requirement during POS analysis and they will be used for modelling the relation with the requirements and interdependency among other verbs. Semantic analysis task utilizes semantic tagger to analyse the context of the phrase in which the verb is used. This information is used to identify verbs used to describe similar requirements. Based on the semantic analysis performed, duplication of the verbs in terms of the context is discarded during filter verbs identification process. Next, we map the requirements using a matrix during map relationship view process. The requirements shared by more than one verb and the scattered verbs are identified based on the relationship view. Finally, identify the dominating verb in the requirement, which are the candidate aspect and model them to identify the crosscutting concern using Action View Model as used in Theme/Doc approach. This paper described a tool that provides automated support for crosscutting concern identification at the requirements level. The tool utilises natural language processing technique to reason about properties of the concerns and model their structure and relationship. But, this model lacks on conflict resolution and implementation and validation of the tool and tests it with case studies.

G. Mussbacher [38] proposed an approach Aspect-oriented User Requirements Notation (AoURN) that extends the User Requirements Notation (URN) with aspects. URN contains two complementary modelling languages Goal-oriented Requirement Language (GRL) and Use Case Maps (UCMs) for goals and scenarios respectively. GRL is a visual modelling notation and supports reasoning about goals and non-functional requirements (NFRs). UCM is a visual scenario notation that supports the definition of scenarios. A scenario describes a specific path through the UCM model where only one alternative at any choice point is taken. Given a scenario definition, a traversal mechanism can highlight the scenario path or transform the scenario into a message sequence chart (MSC). AoURN extends the URN by defining a joinpoint model for the GRL and UCMs. All nodes of GRL graphs or UCMs, which are optional to an actor or a component, are considered as joinpoints. Pointcut expressions are used for matching joinpoints in AoURN models to identify any URN node, and are defined on pointcut diagrams. Pointcut diagrams are standard URN diagrams that allow requirements engineer to increase matching power by using wildcards ("*") and logical expressions (containing "and", "or", and "not"). This approach extends URN with aspects and thus unifies goal-oriented, scenario-based, and aspect-oriented concepts in one framework. Minimal changes to URN ensure that requirements engineers can continue working with goal and scenario models expressed in a familiar notation. At the same time, concerns in goal and scenario models, regardless of whether these concerns crosscut or not, can be managed across model types. But, it uses flexible composition rules that are only limited by the expressiveness of URN itself.

Iqbal and Allen [39] suggest a process modeling approach that represents aspect from the initialization of software to its implementation. It suggests the identification of aspects in the Use Case Model and Sequence Diagrams of the system. Use cases which involve multiple use cases like included or extended use cases may be considered as candidate aspects since they have the probability of crosscutting representation in design as well as in implementation. Similarly, the objects which have communication with multiple objects and which are represented in multiple sequence diagrams may also be regarded as candidate aspects. Proper specification of the candidate aspects can help identifying actual aspects. In this approach, it is not mentioned how to identify aspects. Also, it lacks on implementing the model and validation it with some case studies.

Hamza and Darwish [40] proposed a new approach to identify and model candidate aspects from functional and non-functional requirements of the system, and propagate them to the design phase. The approach needs problem definition as input and produces EBT-NFR analysis model as output. The EBT-NFR model identifies crosscutting NFR and visually shows how they are scattered across the various modules in the system. The process begins with performing requirements and domain analysis. The outcomes of this step are lists of: functional requirements (FRs): non-functional requirements (NFRs), Enduring Business Themes (EBTs), Business Objects (BOs), and Industrial Objects (IOs). The next step is to develop and document main use cases in the system using identified FRs, EBTs, BOs, and IOs. After identifying the main use cases, NFR Matching is performed where each identified NFR in the system is matched with a set of use cases. The next step is Concept Analysis step where, concepts within the problem are discovered using use cases, EBTs, BOs, and IOs. Here, classify the EBTs as concept EBTs (i.e form a formal concept) and none-concept EBT. Next, we need to identify candidate aspects by establishing and understanding the relationships between different NFRs and EBTs in the system. This is achieved by classifying NFRs, and then developing an EBT-NFR model that shows the relationships between EBTs and NFRs. An NFR can be classified into one of three types: Localized-NFR (L-NFR), Distributed-NFR (D-NFR), or an Aspect-EBT (A-EBT). To classify a NFR, calculate Coupling Factor (CF), which measures the coupling between a NFR and each EBT in the system. Finally, develop the EBT-NFR analysis model that visualizes the interaction between the different NFRs and EBT modules in the system and can be used as a link to the design phase. The approach uses Formal Concept Analysis and EBTs that can be identified using Software Stability Model to understand the interaction between NFRs and FRs, and to identify possible aspects in early stages of the development. Further, the proposed approach is the only approach that identifies crosscutting NFRs with respect to the structural nature of the system. The approach lacks a tool to partially automate the proposed approach and applicability of the approach to several case-studies to validate the results.

Liu et al. [41] proposed a use case and non-functional scenario template-based approach to identify aspects. This approach consists of a sequence of activities. The process begins with identifying and defining actors and use cases and building an initial use case model. The next activity is to refine use case model by identifying extensions and inclusions of the use cases. After this, describe NFRs at key association points in the use case model. Association points are of many types as NFRs Association Points, which are specific points of use cases where NFRs can be associated; use case association points associate NFRs to the described functionality, actor association points specify NFRs related to external entities, actor-use case association points represent NFRs related to interaction between external entities and a functionality, and system boundary association points define NFRs that are global in nature. A concern can be identified with a set of architectural policies, and each of these can be described using specific dimensions that specify with more details the NFRs in architectural terms in each use case. Hence, describe architectural policies at platform-independent level through Architectural Policy Scenarios. Finally, identify aspects, where an aspect is a function that influences more functions or more use cases. This approach is based on use cases and described and map non-functional requirements into function and architectures through non-functional scenario template. It not only improves modularity in the requirements which make it possible to begin tackling the problem of tangling, scattering of the requirement as early as in requirement analysis phrase, but also improves traceability from requirement analysis level to implement level, so it achieves a smooth transition between the system analysis and the design. It lacks on supporting the approach with formal method and applying it in more case studies and real systems.

3. A Roadmap to Research

Based on the above discussion, we present a roadmap to our research. The focus of our research work will be exactly on handling crosscutting concerns during requirements phase and to propose a new requirements engineering model for advanced separation of concerns using aspect-oriented concepts having the following challenges:

- Prevent the *tyranny of dominant decomposition* symptom.
- Improve the ability to identify, specify and compose both crosscutting and non-crosscutting concerns.
- Handle conflicts that could emerge when two or more crosscutting concerns affect each other.

4. Conclusion

Crosscutting concerns are the concerns which affect other concerns. These concerns often cannot be clearly decomposed from the rest of the system in both design and implementation and hence result in scattering, tangling, or both; that are difficult to understand and maintain. AOSD is used to identify and specify such crosscutting concerns in separate modules, known as aspects. This results in better support for modularization hence reducing development, maintenance and evolution costs.

AORE is a process that focuses on identifying, analyzing, specifying, verifying, and managing the crosscutting concerns at the early stages of software development. In this literature review, we have discussed many AORE approaches to deal with crosscutting concerns at early stages of software development. As compared with traditional approaches like use cases, viewpoints, and goals; AORE approaches are too young and still need to validate them with more case studies and avoid the tyranny of dominant decomposition, improve the ability for identifying both functional and non-functional crosscutting concerns, offer an automatic mechanism for specifying concern compositions, and to handle conflicts (if any). Some work has been done in this area, but the development of a complete methodology is needed. Without the formulation of this methodology, the full benefits of the aspect-oriented programming paradigm cannot be realized.

References

[1] P. Naur and B. Randell, "*Software Engineering: Report of the Working Conference on Software Engineering*", Garmisch, Germany, October 1968. NATO Science Committee, 1969.
[2] The Standish Group. Chaos Report. Technical report, Standish Group International, 1995, http://www.it-

cortex.com/Stat_Failure_Rate.htm#The%20Chaos%20Report %20(1995).

[3] The Standish Group, “CHAOS Summary 2009”, Technical report, Standish Group International, Boston, Massachusetts, April 23, 2009, http://www1.standishgroup.com/newsroom/chaos_2009.php

[4] Dijkstra, E., “A Discipline of Programming”, 0-13-215871-X, Prentice-Hall, 1976.

[5] D. L. Parnas, “*A Technique for Software Module Specification with Examples*”, Communications of the ACM (CACM), 15(5):330–336, 1972.

[6] D. L. Parnas, “*On the Criteria to be Used in Decomposing Systems into Modules*”, Communications of the ACM (CACM), 15(12):1053–1058, 1972.

[7] Mark.E. et al., “*Do Crosscutting Concerns Cause Defects?*”, IEEE Transactions On Software Engineering, Vol. 34, No. 4, July/August 2008.

[8] ACM, "*Special Issue on Aspect-Oriented Programming*", Communications of the ACM, 44 (10), 2001.

[9] Rashid, A., Moreira, A., Araújo, J., “*Modularization and Composition of Aspectual Requirements*”, In 2nd Aspect-Oriented Software Development Conference (AOSD'03), Boston, USA, ACM Press. 11-20, 2003.

[10] Baniassad, E., Clements, P., Araújo, J., Moreira, A., Rashid, A., Tekinerdogan, B., "*Discovering Early Aspects*", IEEE Software Special Issue on Aspect-Oriented Programming. 23(1): 61-70, 2006.

[11] Suzuki, J., Yamamoto, Y., “*Extending UML with Aspects: Aspect Support in the Design Phase*”, In Object-Oriented Technology Workshop at 13th European Conference on Object-Oriented Programming (ECOOP'99), Lisbon, Portugal, 1999.

[12] Clarke, S., Walker, R., “*Composition Patterns: An Approach to Designing Reusable Aspects*”, In 23rd International Conference on Software Engineering, (ICSE'01), Ontario, Canada, ACM Press, 5-14, 2001.

[13] France, R., Ghosh, S., "*A UML-Based Pattern Specification Technique*", IEEE Transactions on Software Engineering, IEEE Computer Society, **30**(3): 193-207, 2004.

[14] Moreira, A., Araújo, J., Brito, I., “*Crosscutting Quality Attributes for Requirements Engineering*”, In 14th Software Engineering and Knowledge Engineering Conference (SEKE'02), Ischia, Italy, ACM Press. 167 – 174, 2002.

[15] Sutton Jr, S., Rouvellou, I., “*Modeling of Software Concerns in Cosmos*”, In 1st Aspect-Oriented Software Development Conference (AOSD'02), Enschede, Netherlands, ACM, 127-133, 2002.

[16] Rashid, A., Moreira, A., Araújo, J., “*Modularization and Composition of Aspectual Requirements*”, In 2nd Aspect-Oriented Software Development Conference (AOSD'03), Boston, USA, ACM Press, 11-20, 2003.

[17] Rashid A. et al., “*Modularisation and Composition of Aspectual Requirements*”, AOSD 2003, ACM, pp. 11-20, 2003.

[18] Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap. In: Proc. of the IEEE Int. Conf. on Soft. Eng. (ICSE), pp. 35–46 (2000)

[19] Parnas, D.L.: Software engineering programmes are not computer science programmes. Ann. Soft. Eng. 6(1), 19–37 (1999).

[20] Betty H.C. Cheng and Joanne M. Atlee, “*Current and Future Research Directions in Requirements Engineering*”, Design Requirements Workshop, LNBIP 14, pp. 11–43, Springer-Verlag Berlin Heidelberg 2009.

[21] Jacobson, I., Chirsterson, M., Jonsson, P., Overgaard, G., “*Object-Oriented Software Engineering - a Use Case Driven Approach*”. 978-0201544350. Addison-Wesley, 1992.

[22] Sommerville, I., Sawyer, P. (1997b), “*Requirements Engineering - A Good Practice Guide*”, 978-0471974444. John Wiley, 1997.

[23] VanLamsweerde, A. “*Goal-Oriented Requirements Engineering: A Guided Tour”*, In 5th Requirements Engineering Conference (RE'01), Toronto, Canada, IEEE Computer Society. 249 – 262, 2001.

[24] M. Bruntink, A.V. Derusen, R.V. Engelen, T. Tourwe, “*On the Use of Clone Detection for Identifying Crosscutting Concern Code”*, IEEE Transactions on Software Engineering, Vol. 31, No. 10, October 2005.

[25] J. Grundy, "*Aspect-Oriented Requirements Engineering for Component-based Software Systems*", IEEE International Symposium on Requirements Engineering, IEEE CS, pp. 84-91, 1999.

[26] Rashid, A., Sawyer, P., Moreira, A., and Araújo, J. "*Early Aspects: a Model for Aspect-Oriented Requirements Engineering*", Proc. of Int. Conference on Requirements Engineering (RE'02), 2002.

[27] E. Baniassad, S. Clarke, "*Theme: An Approach for Aspect-Oriented Analysis and Design*", In Proceedings of the 26th Int. Conf. on Software Engineering (ICSE04), 2004.

[28] Araújo, J. Whittle, and D-K. Kim, “*Modeling And Composing Scenario-Based Requirements With Aspects*” In Proc. of the 12th IEEE International Requirements Engineering Conference (RE 04), 2004.

[29] Jacobson, I.,” Aspect-Oriented Software Development with Use Cases”, 978-0-321-26888-4, Addison-Wesley, 2004.

[30] A. Moreira, J. Araújo, A. Rashid, “*A Concern-Oriented Requirements Engineering Model*”, *Proc. Conference on Advanced Information Systems Engineering*, Portugal, LNCS 3520, pp. 293 – 308, Springer-Verlag Berlin Heidelberg 2005.

[31] J. Araujo and J. C. Ribeiro “*Towards an Aspect-Oriented Agile Requirements Approach*”, Proceedings of the 2005 Eighth International Workshop on Principles of Software Evolution (IWPSE’05), IEEE 2005.

[32] Isabel Sofia Brito and Ana Moreira, “*Towards an Integrated Approach for Aspectual Requirements*”, 14th IEEE International Requirements Engineering Conference (RE'06), IEEE 2006.

[33] Zhang Jingjun, Li Furong, and Zhang Yang, “*Aspect-Oriented Requirements Modeling*”, Proceeding of the 31st IEEE Software Engineering Workshop SEW-31 (SEW’07), Baltimore, MD, USA, 2007.

[34] Chitchyan, R., Rashid, A., Rayson, P.,Waters, R., “*Semantics-Based Composition for Aspect-Oriented Requirements Engineering*”, In 6th Aspect-Oriented Software Development Conference (AOSD'07), Vancouver, Canada, ACM Press. 36-48, 2007.

[35] Jing Zhang, Yan Liu, Michael Jiang, and John Strassner, “*An Aspect-Oriented Approach to Handling Crosscutting Concerns in Activity Modeling*”, Proceedings of the International MultiConference of Engineers and Computer Scientists 2008 Vol I, IMECS 2008, Hong Kong, 19-21 March, 2008.

[36] Caroline C. Budwell and Frank J. Mitropoulos, "*The SLAI Methodology: An Aspect-Oriented Requirement Identification Process*", 2008 International Conference on Computer Science and Software Engineering, pp. 296-301, 978-0-7695-3336-0/08, IEEE 2008.

[37] Busyairah Syd Ali and Zarinah Mohd. Kasirun,"*Developing Tool for Crosscutting Concern Identification using NLP*", IEEE 2008.

[38] G. Mussbacher, "*Aspect-Oriented User Requirements Notation: Aspects in Goal and Scenario Models*", MoDELS 2007 Workshops, LNCS 5002, pp. 305–316, 2008, Springer-Verlag Berlin Heidelberg 2008.

[39] S. Iqbal, and G. Allen, "*Representing Aspects in Design*", presented at 2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering, Tianjin, China , 2009.

[40] S. Hamza and D. Darwish, "*On the Discovery of Candidate Aspects in Software Requirements*", Proc. Of Sixth International Conference on Information Technology: New Generations, 2009.

[41] Xiaojuan Zheng, Xiaomei Liu, and shulin Liu, "*Use case And Non-functional Scenario Template-Based Approach to Identify Aspects*", Second International Conference on Computer Engineering and Applications, 2010.

Narender Singh is an Assistant Professor in Maharishi Markandeshwar University, Mullana-Ambala, India and pursuing his PhD degree from Department of Computer Science & Applications, Maharshi Dayanand University, Rohtak, India. He has earned his MCA and M.Phil degrees from Department of Computer Science & Applications, Kurukshetra University, Kurukshetra, India in 2006 and 2008 respectively. He has published more than 5 research papers in national & international journals & conference proceedings. His research interest includes Aspect-Oriented Requirements Engineering and Software Product Lines.

Nasib Singh Gill is Professor and Head, Department of Computer Science & Applications, Maharshi Dayanand University, Rohtak, India. He has earned his Doctorate in Computer Science in the year 1996 under the supervision of a renowned academician and researcher – Prof. P.S. Grover of Delhi University and carried out his Post-Doctoral research at Brunel University, West London during 2001-2002. He has received *Commonwealth Fellowship Award* of British Government for the Year 2001. He has published more than 145 research papers in national & international journals, conference proceedings, bulletins, books, and newspapers. He has authored three popular books, namely, *'Software Engineering', 'Digital Design and Computer Organisation' and 'Essentials of Computer and Network Technology'*. He is a fellow of several professional bodies including IETE (The Institution of Electronics and Telecommunication Engineers). He has been awarded with *'Best Paper Award'* by Computer Society of India in the year 1994 for contributing the best paper *"A New Program Complexity Measure"* in their Journal. He is presently guiding researchers in the areas - *Measurement of Component-based Systems, Complexity of Software Systems, Component-based Testing, Data mining & Data warehousing, Aspect-Oriented Software Development, and NLP.*