Chapter 1

INTRODUCTION

"The amateur software engineer is always in search of magic, some sensational method or tool whose application promises to render software development trivial. It is the mark of the professional software engineer to know that no such panacea exists."

- Grady Booch

1.1 Introduction

Over the last few decades, software is evolving at a rapid pace along with more advanced technologies. Developing the complex software system is not a straightforward process but it includes implementation of high level concepts and techniques. A large number of projects fail and result into software crisis [PRE 92] because they violate some constraints. The software crisis means projects running over-time, projects running over-budget, low-quality software, software that did not meet its requirements, projects that were unmanageable, and code that was difficult to maintain. In 1968, the term "Software Engineering" was introduced at NATO Working Conference [NAU 69] on Software Engineering to cope with software crisis.

Edsger Dijkstra, a pioneer of software engineering, explained the major cause for the software crisis as follows [DIJ 72]:

"...machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.”

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 [TSG 95], only about 16% of software projects were successful, 53% were full with problems (cost or budget overruns, content deficiencies), and 31% were cancelled. According to the Standish Group's just-released report, "CHAOS Summary 2009 [TSG 09]" only 32% of all projects succeeding which are delivered on time, on budget, with required features and functions, 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 two fundamental principles that drive the research in software engineering since the early days. Concerns are those interests which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders [IEE 00]. Concerns can be more generally defined as "any matter of interest in a software system" [SUT 02]. Concerns can range from high-level notions like security and quality of service to low-level notions such as caching and buffering. They can be functional, like features or business rules, or nonfunctional (systemic), such as synchronization and transaction management. [HIL 99] has decided to express concerns in the form of questions: "How reliable is this system?", "What function does the system perform?" and "How is the system deployed?" [AKS 01] distinguishes concerns as problem domain concerns and solution domain concerns. Problem domain concerns represent concerns as they are defined from the client perspective. They specifically focus on the functionality of the system as the client expects it. Solution domain concerns represent the concerns as defined by the solution techniques.

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" [DIJ 76]. 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
[PAR 72; PAR 72a] 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. In procedural languages, such as C, FORTRAN, Pascal etc. concerns are encapsulated as functions, procedures or subroutines having various degrees of inter-dependency. In object-oriented programming, the use of classes and inheritance provides a mechanism to achieve this. These programming paradigms have a common way for providing mechanisms for separation of concerns i.e. they used to place concerns into functions, procedures or objects that can be seen as functional units of the system. 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. Some concerns often interact with each other in such a way that they cannot be encapsulated properly within object-oriented constructs. This is because they are too tightly coupled to the behaviors in many other classes or modules. These interactions lead to code tangling - when the elements of code for two concerns are in the same unit and cannot be dissociated - or to code scattering - when a concern involves code spread across several units. Two concerns that are related in such a way that they imply code scattering or code tangling are said to crosscut each other. A concern that crosscuts the main purpose of a unit is a crosscutting concern (with respect to that unit’s decomposition). 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 [MAR 08].

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. Aspect-oriented Software Development (AOSD) [ELR 01] 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 [RAS 03]. 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 [BAN 06].
AOSD was first introduced at programming level, with Aspect-Oriented Programming (AOP), where aspects are handled in code. A number of AOP approaches have been proposed. Work has also been carried out to incorporate aspects, and hence separation of crosscutting concerns, at the design level mainly through extensions to the UML metamodel [SUZ 99; CLA 01; FRA 04]. However, crosscutting concerns are often present before the solution domain, such as in requirements engineering [MOR 02; SUT 02; RAS 03].

Aspect-Oriented Requirements Engineering (AORE) [RAS 03] is an early phase in AOSD that supports separation of crosscutting concerns at requirements level. It not only aims to provide improved separation of crosscutting concerns during requirements engineering, but also to provide a better means to identify and manage conflicts arising due to tangled representations of crosscutting concerns. Since during requirement phase, a more direct contact is maintained with stakeholders [NUS 00], which assists negotiation and decision-making among stakeholders. Such means of early conflict resolution help to establish critical tradeoffs even before the architecture is derived [KHA 05].

A conflict occurs among two or more concerns when one of them tries to vary in a way that is totally opposite i.e. contribute negatively to others [AMR 07]. In AORE, a conflict occurs when two or more crosscutting concerns i.e. aspects having the same priority contribute negatively to each other, need to be composed in the same match point. A match point identifies specific locations in the base concerns where other concerns' behaviour should be composed, or satisfied. Conflict resolution is a process that establishes a critical trade-off among such kind of aspects. So, the conflict resolution process is a compulsory process [SAM 07] and need to achieve it.

Fuzzy logic was invented by Zadeh in 1965 [ZAD 75] for modeling the uncertain and imprecise knowledge in human reasoning. It is suitable for the representation of vague data and concepts on an intuitive basis, such as human linguistic description, e.g. the expressions very important, less important, large, young etc. hence, has proved to be a powerful tool for decision-making in human reasoning, and to handle and manipulate imprecise and noisy data.

Over the last few years, several research efforts have been devoted for handling crosscutting concerns at the early phases of software development especially at
requirements level. These efforts are meaningless unless all the crosscutting concerns are properly identified. Many approaches only consider non-functional concerns as crosscutting concerns. However, crosscutting concerns may also be functional, such as auditing, or validation [MOR 02; RAS 06]. Further, several research efforts have been devoted to resolve conflict in AORE but, still a lot of work is needed. The use of fuzzy intervals to conflict resolution is an emerging area that will incorporate one domain in other.

In this thesis, we have proposed a systematic approach to identify both the functional and non-functional crosscutting concerns during requirements engineering along with its application on a case study. Further, an attempt is made to apply the concept of fuzzy intervals to conflict resolution in AORE which is a new of its kind and that will incorporate one domain in other.

The remaining of this chapter presents the motivation for our work, its aims and objectives, research methodology, scope of the study, outline of the thesis, and list of publications.

1.2 Motivation

*Separation of concerns* and *modularity* are two fundamental principles that drive the research in software engineering since the early days. A lot of techniques exist in literature which follows these fundamental principles of software engineering. Some success in this direction has been achieved. But, still complete separation of concerns is not achieved even in today’s most popular programming paradigm like object-oriented programming. In these traditional techniques, some concerns may be easily encapsulated with their building blocks such as classes, modules, procedures etc. But, same is not possible for another. They are non-modular and spanning over multiple classes, modules, or procedures in a software system and are therefore called *crosscutting concerns*. Common examples of crosscutting concerns are some non-functional requirements such as distribution, synchronization and security. However, functional concerns can also be crosscutting. Therefore, if crosscutting functional or non-functional requirements are hard to isolate within individual modules, and not effectively modularized, it is not possible to reason about their effect on the system or on each other.
The problems that motivate advanced separation of concerns approaches are becoming more pressing and more widely recognized, and computer science has evolved enough that we have a realistic hope of addressing them. The area is young and poses many open research issues and practical challenges.

AOSD [ELR 01] is another step towards achieving advanced separation of concerns and aims at alleviating the problems of scattering and tangling. It aims at addressing crosscutting concerns by providing means for their systematic identification, separation, representation and composition [RAS 03]. Crosscutting concerns are encapsulated in separate modules, known as aspects, so that localization can be promoted. It later uses composition mechanism to weave them with other core modules at loading time, compilation time, or run-time [BAN 06]. This results in better support for modularization hence reducing development, maintenance and evolution costs.

AOSD was first introduced at programming level, where aspects are handled in code. Many aspect-oriented programming approaches have been proposed such as AspectJ, AspectC, AspectC++, JBoss AOP, JAseCo, HyperJ, adaptive programming, and composition filters. A lot of work also has been carried out at the design level mainly through extensions to the UML meta-model e.g. [SUZ 99; CLA 01]. Research on the use of aspects at the requirements engineering stage is still immature. 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. The identified crosscutting concerns may offer valuable insight at later stages of software development such as architecture design and implementation stages etc. Pedigrees of crosscutting concerns may be established throughout the software development lifecycle. Hence, improving the traceability of a wide range of concerns in a software system and facilitating the system’s evolvability. Moreover, identifying crosscutting concerns at the requirement engineering stage may help to reveal the scope of each concern, detect potential conflicts between concerns and support trade-off negotiation and earlier decision making.

The discussion above illustrates the need for AORE approaches. AORE fills the gap left by the traditional requirements engineering approaches by providing systematic
means for the identification, modularization, representation and composition of
crosscutting requirements, both functional and non-functional.

The major motivation factor in the invention of AORE approaches is to remove some
of the flaws encountered in traditional requirements engineering approaches which are
discussed here as:

These approaches suffer from **tyranny of dominant decomposition** symptom i.e. they
are modularized in only one way by considering only one type of concerns such as
use cases, viewpoints and goals at a time. And the other kinds of concerns that do not
align with that modularization result as scattered and tangled modules. For example,
PREview has focused on non-functional concerns. On the other hand, Use Cases have
focused on functional requirements. In contract, Aspect-Oriented approaches, such as
CORE treat all types of concerns equally and consistently. Thus, the first flaw
encountered that AORE addressed is the equal treatment of all types of concerns of
importance simultaneously.

Some traditional approaches have identified non-functional requirements as
crosscutting requirements. But, they do not consider functional requirements as so.
Also, crosscutting requirements are not modularized separately. In contract, Aspect-
Oriented approaches, such as CORE have considered this issue. Thus, the second flaw
encountered that AORE addressed is to identify and characterize the crosscutting
influence for both functional and non-functional requirements and modularize them
separately as aspects.

Over the last few years, several research efforts have been devoted to developing
AORE models that can help in identifying concerns, specifying concerns, composing
concerns, and handling conflicts in the early phase of software development life cycle.
Some success in this direction has been achieved. But, still handling conflicts among
multiple viewpoints or stakeholders is immature and needs more attention with many
research issues.

### 1.3 Aims and Objectives

- Cognition and study of literature related to the area through the light on the
  several research efforts of researchers. Analysis of various models, tools,
• To propose an integrated AORE process model for advanced separation of concerns that could improve the ability to identify and specify both crosscutting (functional and non-functional) and non-crosscutting concerns.

• To propose a systematic approach to handle conflicts in AORE that could emerge when two or more concerns contribute negative to each other.

• To apply the proposed models/approaches in a case study for validation purpose.

• To identify comparison criteria to evaluate proposed approach against other approaches existing in literature.

In order to achieve the objectives, we have framed research problem under this title.

1.4 Research Methodology

• Gather and analyse the detailed and widespread literature in AORE in order to propose an integrated AORE approach for advanced separation of concerns that could improve the ability to identify and specify both functional and non-functional crosscutting concerns.

• Apply the proposed integrated AORE approach for advanced separation of concerns in a case study for providing proofs about its competency.

• Gather and analyse the data generally for AORE and specifically for conflict resolution in AORE to propose a systematic approach to handle conflicts that could emerge when two or more crosscutting concerns contribute negative to each other.

• Apply the proposed conflict resolution approach in a case study for providing proofs about its competency.

• Identify comparison criteria to evaluate proposed approach against other approaches existing in literature.

The sources which will be used from whom data will be gathered during literature study will be articles, books and web references.
1.5 Scope of the Study

This research work presents a systematic approach to identify both the functional and non-functional crosscutting concerns during requirements engineering. Further, a fuzzy interval based approach to conflict resolution in AORE is presented, which is new of its kind and incorporates one domain in other.

It is not feasible to implement the proposed approaches on large projects during this research work. Hence, to measure the efficiency of these approaches, they are implemented on a case study during research work. So, to completely validate, these approaches need to be implemented in software companies on real projects with some minor changes.

1.6 Outline of the Thesis

Brief outline introduction of the thesis is given in the following section. This thesis is structured into six chapters.

Chapter 1: Introduction. A brief introduction to several major concepts used in this thesis is presented in this chapter. The chapter also presents motivation to the research work, aims and objectives, research methodology, scope of the study, and finally outline of the thesis.

Chapter 2: Study and Analysis of Requirements Engineering Approaches. The second chapter provides background and context related to this thesis. It begins with introduction to some core key terms related to requirements engineering and then follows with a brief introduction to the requirements engineering process. Further, it briefly describes some traditional requirements engineering approaches along with some flaws encountered in these approaches. It finishes with a brief introduction to Aspect-Oriented Software Development. The chapter has given rise to publication [1] and [2].

Chapter 3: Review of Aspect-Oriented Requirements Engineering (AORE) Approaches. The third chapter deals with exhaustive literature review. Here, an attempt is made to find out strength as well as shortcomings of each approach by presenting a critical analysis of AORE approaches exists in the literature. Finally, a roadmap to this research work is presented here. The chapter has given rise to publication [1] and [2].
Chapter 4: Towards an Integrated AORE Process Model. This chapter presents an integrated AORE process model, specially focused on identifying crosscutting concerns during requirements engineering. The approach supports the identification of concerns, specification of concerns, and identification of both functional and non-functional crosscutting concerns. This approach also supports conflict resolution that could emerge at requirements level. This chapter illustrates the proposed approach with a case study. This chapter also presents comparison criteria to evaluate proposed approach against other approaches existing in literature. The chapter finishes with a summary. The chapter has given rise to publication [2] and [3].

Chapter 5: Fuzzy Interval Based Approach for Handling Conflicts in AORE. The use of fuzzy logic to conflict resolution is an emerging area that will incorporate one domain in other. In this chapter, an attempt is made to apply the concept of fuzzy intervals to conflict resolution in AORE. The first section introduces the conflict resolution in AORE. Further, it illustrates the proposed approach with a case study. The chapter finishes with a summary. The chapter has given rise to publication [4].

Chapter 6: Conclusions and Future Work. This chapter is divided into four sections. The first section presents background of the thesis. The second section presents significant contribution made in the thesis. Third section presents the future works and finally the chapter finishes with concluding remarks.