Chapter 1

Introduction

Software designing is an art. To enhance the designing capability, software researchers and developers have been creating abstractions that focus on developing software in terms of design rather than creating the underlying computing environment. Designing is required not only for the initial development process of software but also for software evolution which is a continuous process for adaption, extension and customization of software.

Over the last few years, a new approach has developed to define a software solution’s architecture known as Model Driven Development (MDD). MDD gives architects the ability to define and communicate a solution while creating artifacts that become part of the overall solution [34]. The increased complexity of software products, their brief development cycles and a need to develop reusable components of software have imposed challenge to MDD. In order to meet the above mentioned challenges, there is a need to improve the handling of MDD technologies. Moreover, MDD has a bright future in bringing revolution in large-scale software development.

Different approaches have been developed to raise the level of abstraction in context of software development. Object-Oriented Programming is one of the approaches and a very popular concept of modularizing software. The decomposition of software is in terms of objects that abstract behavior and data in a single unit. It encourages software reuse by providing design and language constructs for modularity, encapsulation, inheritance and polymorphism. Although OOP has brought a phenomenal change in abstractions and modularity mechanisms but the object-oriented modules are not enough to cope up with certain characteristics like cross-cutting concerns. A major problem faced by the software developers is maintaining the code which becomes very
difficult in cleanly separating concerns in modules. An attempt to make minor changes in the program design may require several updates to a large number of unrelated modules. This major drawback of OOP has been the main motivation of a new programming paradigm known as Aspect-Oriented Programming (AOP).

AOP is a new concept that separates crosscutting concerns into single units called aspects. An aspect is a modular unit of crosscutting implementation. It encapsulates behaviors that affect multiple classes into reusable modules.

Crosscutting concerns are more prominent in large-scale software products. Feature-Oriented Software Development (FOSD) is another new concept which enables the construction, customization and synthesis of large, medium and small scale software products efficiently. Feature-Oriented Programming (FOP) is a programming paradigm, in which a feature is a unit of functionality of a software system that satisfies a requirement, represents a design decision and provides a potential configuration option [14]. From a set of features, many different software systems can be generated that share some common features and differ in other features. The set of software systems that are generated from a set of features is also called a Software Product Line (SPL) [32, 33].

SPL can be explored in a better way with the integration of AOP and FOP. It has been found that AOP and FOP are complementary technologies that can be combined to overcome their individual limitations [98]. In order to unify AOP and FOP it has to be visualized in context of architectural meta-programming [18].

Metaprogramming and architectural metaprogramming are two major contributions towards Model Driven Development which are used individually in software automation. It has been felt that, architectural meta-programming principles can be applied to any metaprogramming approach. This may reveal the fact that, architectural metaprogramming is mandatory in the design phase of metaprogramming. It has been also observed that, the existing metaprogramming approaches are very limited. Object-Oriented Programming uses a metaprogramming facility known as template metaprogramming that is available in C++. Though the technique is very powerful, but the complexity of the syntax etc. has not made it widely acceptable. As a consequence,
code maintenance and re-use of the existing codes cannot be efficiently obtained through template metaprogramming. OOP, on the other hand, cannot handle the crosscutting concerns which AOP can handle.

In order to avoid the complexity of template metaprogramming, to make metaprogramming popular as well as user-friendly in addition to integration of AOP and FOP in metaprogramming, a metaprogramming approach has been developed in a compositional language. It is proposed in this work to develop a formal framework which shall enable the programmer to define his own derived compositional operators. These definitions can exploit the full expressive power of the underlying computational language by SPL. This is the motivation to design and implement a SPL through metaprogramming. The proposed elementary software product line is known as Array Product Line (APL) which is a metaprogramming approach to implement SPL. It is desired to illustrate architectural metaprogramming using software product line, by first mentioning the principles and then by justifying that the present work on APL is designed on the basis of architectural metaprogramming.

The application of the principles of architectural metaprogramming in the present APL proves that AOP, FOP and refactoring are all part of MDD. On the other hand, the metaexpression generation and obtaining the codes from metaexpressions through a GUI interface is a user-friendly and better technique for Model Driven Development as compared to the existing one.

1.1 Scope of the Thesis

This thesis focuses on the following domains of Model Driven Development:

(a) Aspect-Oriented and Feature-Oriented Programming
(b) Modular Refactoring
(c) Meta-programming and Meta-expressions

1.1.1 Aspect-Oriented and Feature-Oriented Programming

Aspect-Oriented Programming and Feature-Oriented Programming are two new paradigms of software modularization where the former modularizes crosscutting
concerns and the later implements software product lines through modular units called features. It is desired to provide better modularizations techniques by presenting an elementary software product line named as Array Product Line (APL) which applies AOP and FOP. AOP encapsulates the crosscutting concerns like login etc. FOP defines constants and functions and is composed to define a domain of programs that can be synthesized. It is also visualized that AOP and FOP as complementary techniques.

1.1.2 Modular Refactoring

Refactoring is a technique of restructuring the software without changing the behavior in order to improve the quality of the software. Model driven refactoring is known as modular refactoring which changes the modular structures either by adding features or by removing features from the models. Since refactoring is a horizontal transformation, a technique known as Refactoring Flow Diagram (RFD) is presented, which detects impure refactoring in model level and in code level. The advantages of RFD are as follows:

- depicting the different processes involved in refactoring
- sequencing the processes along with tracing the impure refactoring

If this approach is incorporated in a tool then it will help in early detection of impure refactoring.

1.1.3 Metaprogramming and Metaexpressions

Metaprogramming, the art of programming programs that read, transform or write other programs, appears naturally in the chain of software development [38]. In this thesis, automated program construction has been done through software product line (Array Product Line) by generating metaexpressions and by converting the expression to source code. The metaexpressions are generated by FOP domain model which is an innovative approach in Model Driven Architecture for enhancing the automation process in MDD.
1.2 Organization of the Thesis

Model Driven Development of software has several approaches which are required to be analyzed for finding some efficient techniques. This research work will be of help to those who want to find better techniques in MDD. The present work has been conducted in this novel area and has been organized in the following ways:

This research work comprises of total ten Chapters, out of which seven chapters are contributory research work.

- The literature survey is done in Chapter 2 where the details about Model Driven Development, Aspect-Oriented Programming, Feature-Oriented Programming and Refactoring have been studied and presented. The problem statement has been formulated and the motivation for this research work has been stated in this chapter.

- Chapter 3 presents the role of AOP in MDD. In this chapter we propose for a login module where we encapsulated some concerns related to AOP. A case study on Smart Telephone Network Product Line (STNPL) has been introduced to justify the significance of login process in any product line.

- In Chapter 4 we discussed about FOP in context of better reusability and better way to handle FOP. We demonstrated FOP with a basic software product line and also mentioned about the significance of metaprograms and transformations in FOP. As a part of STNPL case study, we illustrated FOP concepts in STNPL.

- Chapter 5 illustrates the similarities of FOP and AOP in order to justify that these two approaches complement each other. This is achieved by integrating the AOP modules with FOP modules both for software product line and STNPL.

- In Chapter 6 the early detection of impure refactoring has been explained by introducing the RFD concepts.

- The modular refactoring in code level and design level is given in Chapter 7. The basic purpose of this chapter is to apply our proposed mechanisms of detecting impure refactoring in the login module i.e. the software product line and in STNPL, to justify its significance.
An elementary software product line has been designed through activity diagrams and flow diagrams in Chapter 8.

In Chapter 9 the proposed SPL user interface is designed, metaexpressions are obtained and finally the source code generation from metaexpression has been done.

Chapter 10 concludes the overall thesis followed by references.