Chapter 10

Conclusions & Future Works

This thesis, has explored AOP, FOP and refactoring in two different dimensions from the programming level and another from the architectural metaprogramming level. The proposed login module in AOP can be integrated to any software for identification of an authentic user. In order to implement the login module, Software Product Line which is named as Array Product Line (APL) has been proposed. This is a miniature product line which has been designed to support that, features are required in software product line and it is used in an incremental way to automate software designing. The designed APL not only generates metaexpressions but also source code to any preferred high level language like C/C++/Java. Modular refactoring has been done in the login model and in the designed user interface. Apart from this, the case study of Smart Telephone Network Product Line shows the necessity of AOP, FOP and refactoring in MDD for any product line. The designing and functioning of APL and STNPL show that AOP, FOP, MDD and refactoring are not different areas but are all are complementary to each other.

10.1 Conclusions

The introductory chapter starts with a brief introduction of AOP, FOP, refactoring, MDD and metaprogramming in order to present an overview of the available MDD techniques. It also covers the objective and the scope of the thesis which is followed by the organization of the thesis.

Chapter 2 comprises an in depth discussion of Aspect-Oriented Programming, Feature-Oriented Programming, Model Driven Development and Refactoring. All existing concepts and practices are studied in these areas, in order to focus on the latest
developments in MDD techniques, which have been presented in this chapter. Finally, the motivation and problem statement have been stated.

Chapter 3 to Chapter 9 comprises technical contributions. In Chapter 3, the crosscutting concerns are discussed, few of them are identified and their impacts in software development have been analyzed. As a result, a security module scheme has been proposed. The proposed model recognizes two categories of users - developers, who have an access to the code manipulation and end users for using the software. The core concerns and non-functional concerns have been identified. The class and state diagrams have been used for composing the concerns which have been further refactored to AOP in order to encapsulate crosscutting concerns. This login module is a generalized module as it can be integrated to any software as an access control mechanism. A case study on Smart Telephone Network Product Line has been introduced in this chapter which uses an AOP approach to design a security aspect by validating the user’s identity.

Chapter 4 has focused on the steps that are required for handling FOP by consulting the different definitions of features. The lifter mechanism (which is a generalization of inheritance), has been supported through an example of different Array operations. The lifters contain array interfaces which inherit methods like insert, traverse etc. The example is a pseudo code which depicts a feature handling mechanism. This lifter mechanism can be incorporated in language level as it allows composing objects with individual features. If languages like FeatureC++ provide this facility then designing features with individual services become very easy. This characteristic of FOP will be a very powerful tool in MDD. Similar features are designed for STNPL to facilitate the call operation for a genuine user.

Chapter 5 deals with the unification of AOP and FOP. Aspectual Mixin Layers are used to integrate AOP and FOP in the proposed security login and in the designing of STNPL. The designing of the extended aspectual layers provides more flexibility in the logging process for any security module. Moreover, the extended version confirms that AOP and FOP can complement each other by using them in a layered structure. This has further helped in designing the software product line in context of variability.
management that is essential for identification, design, implementation and tracing of flexibility in software product lines (SPLs). Structural mapping has also been adopted to express variability management. Finally, a feature model of the login process has been produced. It has been observed that structural mapping can be used in a better way if some supporting tools are given.

Refactoring and model based refactoring have been discussed in Chapter 6. This discussion has been made in context of classification and techniques of refactoring. The main objective has been to detect impure refactoring at early state. The graph search and the graph transform techniques have been analysed to detect impure refactorings and a method named Refactoring Flow Diagram (RFD) has been proposed which incorporates the graph transforming technique. RFD can come in several levels starting from Level 0. This concept has been used by application of RFD in certain cases where the refactorings were pure. This has been done by producing RFD in several levels in order to validate the theory that RFD can detect impure refactoring at an early state. RFD has also been used in the case study to demonstrate its necessity in designing phase of any product line.

The modular refactoring is explored in Chapter 7 by using UML and OCL constraints. The class diagrams have been refactored for implementing modular refactoring. The scope of AOP refactoring has also been investigated which has been followed by refactoring an existing login aspect. The class diagrams refactoring of array handling are platform independent refactoring whereas the refactoring of the class diagrams of the Security login (as proposed in Chapter 3) to aspect code is a platform dependent refactoring. The RFDs have been applied at different levels, on the security login refactoring resulting in a pure refactoring. As a part of the modular refactoring, class diagrams of STNPL have been refactored. This indicates that modular refactoring is an essential part of MDD.

In Chapter 8, a Software product Line named as Array Product Line has been designed that works on a metaprogramming approach. The user’s responses are taken in a form of metaexpressions, which on demand can be
converted to source code. The designing of the user interfaces, metaexpression generation etc. have been done through activity diagrams and through flow diagrams which are refactored later on by applying the modular refactoring. It has been observed that, activity diagrams, flow diagrams, refactoring on such diagrams (i.e. in models) etc. must be the first stage of designing prior to code development. The activity diagrams and the flow diagrams present a holistic view of system design. This has been further demonstrated in the flow diagrams of STNPL. This modular approach enables a better designing of an evolutionary system.

Chapter 9 deals with source code generation of the desired metaexpression either in an object oriented program (C++/Java) or in a procedure oriented program (C), as preferred by the user. It has been designed in language level and also in architectural model using GenVoca. The later helps in application of the principles of architectural metaprogramming in APL code to justify the validity. It has been found that all the principles are at par with the designed source code segments of APL. This is a major achievement as it supports the fact that AOP, FOP and refactoring are all Model Driven Development approaches where MDD is an architectural programming paradigm. Architectural metaprogramming can be explored in different dimensions in order to enhance the capability of MDD. The metaexpressions that have been obtained by selecting features and developing source code for the metaexpression are cases of software automation that has been achieved by a new method in MDD. Hence, software automation can also be achieved by first generating the metaexpression through feature selections and then converting the metaexpression to the source code. This technique of software automation can add a new feather to MDD approach.

10.2 Future Works

For future work, it is recommended that lifter concepts may be incorporated in high level language like FeatureC++. This will allow easy and flexible composition of feature in SPL which will help in automated software generation, since it is a major need in MDD. Recommendation is also made for use of graph transformation in RFD as
a tool to make an early detection of impure refactoring. This will make refactoring popular in both code and model level.

Apart from the above mentioned suggestions, development of a product line which can cater to any linear or non-linear data structure is also recommended. This is an extension of the proposed APL. Such a data structure can be used as a tool for handling and analyzing linear or non-linear data structures like stack, queue, linked list, tree, graph etc. The MDD techniques presented in this work can be used for development of the new product line.