International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-4 Number-2 Issue-15 June-2014

Model Driven Development: Research Issues and Opportunities

Mahua Banerjee[1], Sushil Ranjan Roy[2], Satya Narayan Singh[3]

Abstract

Software engineering aims at techniques for producing better software products with limited resources. This encourages the reuse of the existing resources which are mostly available in terms of modules. Model Driven Development (MDD) is an approach which facilitates the reuse of existing models. Moreover, models are developed not only with high level of abstraction but also with better provision of reusability. This further motivates software developers and engineers for software automation. MDD has shown a significant role in software automation which can be further improved by incorporating the latest approach of software development like Aspect-Oriented and Feature-Oriented Programming.

Keywords

Aspect-Oriented Programming, Feature-Oriented Programming, Model Driven Development, Program families

1. Introduction

One of the primary goals of software engineering is to assist developers in dealing with this dynamic environment, where a large part of work is to change existing code rather than to write new code. The way in which this can be done is through better separation of concerns. Programming languages provide mechanisms for dividing programs into modules that represent particular design decisions, features or pieces of functionality which can be generally referred to as concerns. Modularizing concerns help during evolution because developers do not have to deal with the entire program every time they want to make a change. They can focus on just the modules that relate to their task and ensures that the implementation of the revised modules will not affect the other modules by behaviour preservation of the entire software. Modules also provide a means of encapsulating generic functionality so that code can be reused across multiple projects [1].

However, not all concerns can be easily modularized. When a designer chooses a decomposition of a program into modules, he/she does so with the intent of making the expected evolution task easier for developers to perform. In practice, finding a decomposition that supports all evolution tasks is often impossible. In some cases this is due to new requirements or due to environmental changes that could not have been predicted and so were hence not planned for. In other cases, the programming language used to implement the software does not provide adequate means to encapsulate the concerns neatly. This leads to the existence of concerns whose implementations are scattered across multiple modules. This mismatch between the chosen decomposition and the required programming tasks is often referred to as the tyranny of the dominant decomposition and is one of the main motivations for aspect-oriented programming (AOP) [2].

2. Aspect-oriented programming

Aspect-oriented programming builds on top of Object-Oriented Programming (OOP) by introducing new forms of modularity. AOP languages provide more flexibility in choosing decomposition through the use of aspects which define both state and behaviour that can be woven into the object-oriented structure of a program [3]. AOP approaches make easier to modularize concerns that were previously scattered amongst object-oriented classes. At the same time they tend to scatter the implementation of classes across aspects. This makes some tasks easier to perform at the expense of making others more difficult.

Concerns and AOP Concepts

Object-Oriented Programming (OOP) is probably the most commonly used programming paradigm today. Functional, procedural and object-oriented programming languages have a common way of abstracting and separating out concerns in the sense that they rely on explicitly calling subprograms (subroutines, procedure, methods, etc.) that represent

Manuscript received June 16, 2014.
Mahua Banerjee, Department of Information Technology, Xavier Institute of Social Service, Ranchi, India.
Sushil Ranjan Roy, Department of Information Technology, Xavier Institute of Social Service, Ranchi, India.
Satya Narayan Singh, Department of Information Technology, Xavier Institute of Social Service, Ranchi, India.

functional units of the system. However, all concerns cannot be encapsulated properly in a functional decomposition. As a result, they must be coordinated with other functional units and they usually involve code scattered throughout several of these functional units. Aspect-Oriented Programming aims at better separation of concerns by providing the aspect as a means to encapsulate such crosscutting concerns [4]. Aspects are capable of controlling the scattering and tangling of codes. It is obtained by specifying places in the program's execution (known as joinpoints) where certain codes (called advice) are executed. This is the main aim of Aspect-Oriented Programming, where concerns which cut across each other can be linked together, and yet be encapsulated transparently as separate program entities. The general style of programming that arises out of this aim consists of program statements of the following form:

"An action X is performed, whenever a potential condition arises, in the programs". This implies that aspects are modular units which encode statements and are executed on some pre-defined conditions. On account of pre-defined conditions, the aspect is implemented before, after or during a certain event, in order to trace certain activities. This is similar to database triggering facility which is required for the security of a database. Aspect modules can be triggered for security activities of any software. In most AOP languages the concept of an aspect extends the concept of a class. Aspects can contain members similar to members of a class, i.e., aspects can contain methods, fields, or inner classes and interfaces [8]. Besides structural elements known from OOP, (e.g., methods and fields), aspects may contain also join points, pointcuts, advice and inter-type declarations [5-7].

(a) Join point: A *join point* is a point in the structure or in the execution of a program where a concern crosscutting that part of the program might intervene. The ability of an AOP language to support crosscutting concerns lies in join point. It consists of body of a method called lexical join points, call of particular method and run the code at the required time [19]. The time the code to be executed can be expressed as potential conditions in programs. Join points can also be seen as hooks in a program where other program parts can be conditionally attached and executed. The primary mechanism of AOP is the extension of events occurring at runtime, so-called join points. The static representation of a runtime event is called join point shadow. Join point shadows are for example statements of method calls, object creation, or member access.

(b) Pointcut: A *pointcut* is a subset of all possible join points. The expression of a pointcut is the *pointcut descriptor* (often, the term "pointcut" is used in place of "pointcut descriptor"). A pointcut descriptor defines the potential condition in the above formulation. This condition matches a subset of join points which is the pointcut. In other way a pointcut is a declarative specification of the join points that an aspect will be woven into, i.e., it is an expression (quantification) that determines whether a given join point matches or not.

(c) Advice: The piece of code A (say) that is to be executed when the potential condition arises (i.e. at a join point of the pointcut) is called the *advice*. An advice is a method-like element of an aspect that encapsulates the instructions that are supposed to be executed at a set of join points. Pieces of advice are bound to pointcuts that define the set of join points being advised.

(d) Aspect: The unit of code that defines the pointcuts and the advice related to the same concern is called the *aspect*. An aspect can also be more generally defined as a unit that encapsulates a crosscutting concern.

(e) Inter Type Declaration: Inter type declarations (ITD) are methods or fields that are inserted into OOP classes by an aspect and thus become members of these classes. Additionally, interfaces can be extended with methods and fields. Inter-type declarations are also known as introductions as they inject new members into classes.

The aspect weaver software first integrates the source code of non-crosscutting concerns and the source code of crosscutting concerns into a single unit and then the compiler converts them into object programs and into executable form.

As suggested by Kiczales et al. [9] mechanisms for aspect orientation rest on three pillars:

- a model of the behaviors that can be recognized and exploited (the joinpoints),
- a means of characterizing a subset of these possible behaviors (the ability to define pointcuts),

- a means of implementing the behavior defined in the aspects at the place and at the time when the expected behavior defined in the pointcuts happens (the weaving of the advice).

These three pillars are the "*three critical elements that AO languages have*".

Relevance of AOP in Software Development

The knowledge of AOP concepts and the critical elements of AO languages are not the only requirements for implementing AOP in software development. The software development process, using AOP, needs to identify core functions and crosscutting functions. Core functions deal with designing of required software and crosscutting functions deal with the code designing of scattered and tangled codes. The approach of handling software development using AOP is different from traditional approaches. The major steps required for software development using AOP are the following:

(a) The isolation of functional (non-crosscutting) and non-functional (crosscutting) concerns in the designing phase.
(b) Development of the base program in a conventional programming language, which comprises the non-crosscutting concerns.
(c) Designing aspects, in an aspect-oriented programming language by encapsulating the crosscutting concerns into it.
(d) Finally, aspects are woven into the base program with aspect weaver.

The main advantage of using AOP is that it can be integrated in some conventional languages like, C++, Java etc. As a result, application software like AspectC++, AspectJ etc. [20] have evolved. Though the language level support is available but still AOP approach is not yet popular for the following reasons:

(a) It is mainly used in large-scale application software development.
(b) There is a lack of knowledge in identification/selection of crosscutting concerns.
(c) There is ignorance of the processes of integration of crosscutting concerns and non-crosscutting concerns.
(d) People are unaware of the benefits of using separate modules for crosscutting and non-crosscutting concerns.

In order to cope up with the above mentioned issues it is aspired to promote AOP in software development by introducing the following:

(a) Application of AOP in small-scale software development.
(b) Identification/Selection of crosscutting concerns.
(c) Integration of the crosscutting and non-crosscutting concerns.
(d) Analysis of the benefits of using separate modules for crosscutting and non-crosscutting concerns.

Examples demonstrating the concepts of AOP

An AOP language comprises of the terms like aspect, advice, pointcut, joinpoint etc. In order to have a better understanding of the terms of AOP, the following code fragments are considered.

```
aspect Mymessageaspect
{
  advice call("% %Myclass::Func(...)") : before()
  {
  cout << "I am calling Myclass::Func"<< JoinPoint
:: signature() endl;
   }
}
```

Figure 1: gives a code segment of AOP. The aspect

module name is Mymessageaspect. As stated in section *A*, members like advice and joinpoint are encapsulated in the aspect module to minimize the scattering and tangling of the codes. The encapsulated advice displays the message before any call to Myclass:: Func. The two % used are the wild cards. The first % implies any return type of the function and the second % is to denote that advice execution will be done for those classes, where the name of the class ends with Myclass. The special character (…) matches any number of parameters in the function named Func.

The message will be triggered to each point of the program where the class name ends with Myclass. These points in the program code are known as joinpoints. Set of such joinpoints (all the places where the message is triggered) is pointcut.

The example illustrates before advice, in which the message will be generated only when there is a call to a function Func, with any number of parameters. In before advice, message comes first followed by the function call. This sequence can be reversed by using

after advice. An around advice, will explicitly trigger the joinpoint codes, during the execution of a process.

The advice like before, after etc. can have parameters which can be made available in the form of values in codes of the advice. To demonstrate this, the following example is considered:

```
pointcut spl_user(const char *name) =
        execution("void login(...)") &&
args(name);
advice spl_user(name) : before(const char *name)
{
   cout<< "User "<< name << "is logging in"<<
endl;
}
```

Figure 2: An example of AOP with parametric advice

The pointcut named *spl_user* is given a prototype declaration with initialization to the call of login() which includes a context variable *name* that is bound to it. Hence for every joinpoint, referred by the pointcut *spl_user*, a value of type const *char is provided. This value of each joinpoint gets bind to the code of the advice (i.e. substituted in the advice code). The example demonstrates the accessibility of the actual argument values of a function call, in the advice codes.

3. Feature-oriented programming

Feature-Oriented Programming is a design methodology and a tool for program synthesis. It not only designs a target program declaratively by providing the features it offers but also provides an efficient implementation of features. Product lines are developed by using FOP in widely varying domains like compilers for extensible Java dialects, network protocols, program verification tools etc.

The idea of program families has evolved in order to overcome the software crisis. Instead of designing individual single program a program family is designed which consists of similar programs. The advantage of it is that a developer creates a program by choosing from a set of features. Usually many combinations of features are allowed. This results in a variety of programs. In order to implement product lines, FOP is used. Object-oriented programming had a great success in software development by incorporating modularity through data abstraction and through data hiding. The reusability and flexibility were lacking in object-oriented technique which are essentially the major requirements of software development. Though classes, which are the traditional units of organization of object-oriented software, bring modularity, they fail to develop software in an incremental way. Product lines on the other hand do it. In order to overcome this, FOP can be used to develop modular system product lines. FOP decomposes software into features which are increments in program functionality as they are applied to a program in an incremental fashion. This potentiality improves modularity along with the reusability and flexibility of product lines [10, 11].

Feature modules

Feature modules are distinct code units. Features are not implemented through one single class but are implemented through different classes. To add a feature subsequently means to introduction of code into existing classes. Codes of different classes associated to one feature are merged into one feature module. Feature modules refine other feature modules in a stepwise manner by superimposing the feature modules already assembled.

The specification, modularization and composition of features are provided by some feature-oriented programming languages and tools like AHEAD, Caesar, Feature House, Feature C++ etc. All languages and tools implement feature by feature module. New structures such as classes and methods are introduced when feature modules are added to a base program. This refines the existing ones such as extending methods.

Implementation of features and feature modules

Feature-Oriented Software Development is the process of developing software systems in terms of features. It deals with the study of feature modularity, tools and design techniques that support feature based program synthesis. There are several approaches like GenVoca [12], mixin layers [13], AHEAD [14] etc. which concentrate on encapsulating features as increments over an existing base program, together with a mechanism for combining different features on demand.

(a) GenVoca [12]: GenVoca is a meld of the names Genesis and Avoca. This is a compositional paradigm for defining programs of a product line. It is a tool for defining code constructs in a higher level than in a

program code. In GenVoca, a module is specified by declaring the set of layers that make it up, where each layer defines the aspect of the module. When a layer is added, it adds methods and/or arguments to existing methods. Layers can be mixed and matched in very flexible way. GenVoca creates C++ code.

(b)Mixin Layers [13]: Mixin layer is another approach of implementing features in a layered object oriented fashion and is often known as collaboration based design. Features are implemented by collaborations. Collaborations are collection of roles (classes). A mixin layer is a module that encapsulates fragments of several different classes (roles) so that all fragments are composed consistently. In order to encapsulate fragments of several different classes, mixin-based inheritances are used. Mixins are used for expressing and refining collaborations of classes.

(c) AHEAD [14]: Algebraic Hierarchical Equations for Application Design (AHEAD) supports a hierarchical structure in which a class is a set of methods. A set of methods makes a module and a set of modules makes a software system. The idea of AHEAD model is to decompose programs into separate modular units (features) and to compose stacks of features to derive a concrete program. AHEAD proposes compositional programming. It generalizes the concept for features and feature refinements. The programming in AHEAD style is supported by a set of tools provided by the AHEAD Tool Suite. But the AHEAD Tool Suite is not popular because most of the functionality is provided by command line tools. Hence IDE support for program families was suggested by Lich et al. [15].

All approaches like GenVoca, mixin layers and AHEAD is related to one another. GenVoca features were originally implemented using C preprocessor techniques. Mixin layers, show the connection of features to object-oriented collaboration-based designs. So, we can say that mixin layers are a more advanced version of GenVoca. GenVoca is also related to AHEAD. AHEAD has generalized GenVoca in two ways. Firstly, the internal structure of GenVoca values is presented as tuples. Each program has multiple representations in terms of source, documentation, bytecode etc. A GenVoca value is a tuple of program representations. Each program representation may have sub-representations. A sub-representation may have subordinate sub-representation and so on recursively. In general, a GenVoca value is a tuple of nested tuples that define a hierarchy of representations for a particular program. Secondly, AHEAD expresses features as nested tuples of unary functions called deltas. Deltas can be program refinements (semantics – preserving transformations), extensions (semantics – extending transformations) or interactions (semantics – altering transformations).

Role of fop in modular software development

AHEAD, GenVoca and mixin layers not only deal with the implementation of features and feature modules but also they are all based on model-driven architecture. A model-driven architecture is required for promotion of software automation. The present software automation deals mostly on template metaprogramming. Due to the complexity of template programming, it has not yet been widely accepted for software automation. This motivates, for a better approach of software automation. As a result, higher level of abstractions of programs evolved of which metaprogramming is one of them. The main idea of metaprogramming is that, programs are values and functions transform the values which finally results into the required software.

These metaprogramming techniques can be used to formulate metaexpressions. These metaexpressions can further be treated to develop the software. The present FOP approaches do not directly implement the metaprogramming techniques. However, generations of metaexpressions are similar to GenVoca and AHEAD approach. The former designs the layers in modules and the later designs the layers in algebraic equations. On the other hand, both deal in FOP approach with variations in designing the feature modules. The major issues of any FOP are the following:

(a) Designing the feature modules at the phase of requirement analysis.
(b) More elaborate designing of feature modules, in terms of structure and behaviour of features and their interactions.
(c) Implementing feature modules through language and tools.

To cope up with the above mentioned issues, the following things are required:

(a) The traditional, requirement analysis must be replaced with the first stage of designing of the features. This will make the requirement analysis more logical as the feature modules will reflect the requirements

more distinctly. Different feature designing also becomes easy because the first stage of feature designing is done here.

(b) The feature interactions facilitate feature compositions. This demands appropriate combinations of features.

(c) User friendly tools and language support is required for implementation of the modules.

(d) Technical improvements in FOP are required for software automation.

The above requirements impose the following challenges:

(a) The existing FOP approaches like GenVoca, AHEAD (algebraic model) etc. has not yet been successful in automated software development. It is a step closer to automation of software [21].

(b) Adequate language and tool support is not available. FeatureC++. Feature House etc. provide some of the mechanisms of FOP.

In order to meet the above mentioned challenges, the Model Driven Development techniques are needed to be explored which can be refined further for software automation.

4. Model Driven Development

Model Driven Development (MDD) [16] is a promising area of software development in near future. MDD shifts software development from a code-centric activity to model-centric activity. In order to accomplish this shift, modelling concepts are required at different level of abstractions. Finally the abstract models are transformed to code generic model. MDD supports the use of Domain Specific Languages (DSL). Automation and data exchange methods can be further improved by Model Driven Development. It is a way to define a software solution's architecture. MDD gives the architect an ability to define and communicate a solution which finally becomes a part of the overall solution.

The following tasks are facilitated by a good MDD tool:

(i) It communicates the solution to stakeholders who are not in the development team.

(ii) It helps to interact and facilitate the team that is developing the solution.

(iii) It aids in tracking the history of the decisions behind the solution's design.

MDD technologies

There are many MDD technologies. OMG's Model Driven Architecture (MDA) is one of the popular technologies. In MDA, models are defined in terms of Unified Modeling Language (UML) and are manipulated by graph transformations [17]. The main aim of MDA is to increase productivity and to reuse models through separation of concern and abstraction. In software development process, MDA helps in efficient use of system models and also supports reusing models when creating families of system. According to the definition of Object Management Group (OMG), MDA is a way to organize and manage enterprise architectures that are supported by automated tools and services for both defining models and for facilitating transformations between different model types.

OMG has also formulated the following four principles of MDA:

(a) Models expressed in a well-defined notation are a cornerstone to understanding systems for enterprise-scale solutions.

(b) The building of systems can be organized around a set of modules by imposing a series of transformations between models, organized into an architectural framework of layers and transformations.

(c) A formal underpinning for describing models in a set of meta-models facilitates meaningful integration and transformation among models, and is the basis for automation through tools.

(d) Acceptance and broad adoption of this model-based approach requires industry standards to provide openness to consumers, and foster competition among vendors.

Based on the above principles, OMG identifies four types of models:

(a) Conceptual Independent Model (CIM)
(b) Platform Independent Model (PIM)
(c) Platform Specific Model (PSM)
(d) Implementation Specific Model (ISM)

The transformations from one model to another model can be performed by several MDA Tools, namely, IBM Rational Rose Technical Developer or IBM Rational XDE. The former, transforms a model from UML to executable code in a single step whereas the later transforms an initial analysis model to executable codes in several steps.

IBM has taken a leading role in support for modelling, model driven development and in Model

driven architecture. UML has been defined by IBM which made it (UML) acceptable for the architecting of large-scale software systems. The popularity of MDA is due to the strong support of IBM for OMG. Hence, it is necessary to make an analysis of the present MDD technologies.

Overview of present MDD Techniques

As modelling has become an essential part of software development, knowledge of present MDD techniques are required. MDD approaches provide a better understanding of the system because a system can be analysed through different perspectives like requirement perspectives, analysis perspectives etc. Hence software can be elucidated in multiple views through different models. Also, UML is also not an exception in context of depicting the software in multiple views. This popular graphical modelling language helps in developing, understanding and analysing the different views through models.

UML is primarily used in modern object-oriented modelling. Use case modelling, static modelling, state machine modelling and object interaction modelling are used for the following [22]:

Use Cases: Functional requirements

Static Modelling: Structural view of the system

State Machine Modelling: Behavioural view of the system

Object Interaction Modelling: How objects communicate to each other to realize the use case.

In order to provide the above mentioned facilities in UML for real-time, concurrent and distributed software design methods, the existing techniques are explored. The observations are given in Table 1.

Table1: Analysis of existing MDD Tools

System Type	MDD Tool	Purpose/Full Form	Specification
Real-Time system	CODARTS	Concurrent Design Approach for Real Time Systems	Refinement on existing concurrent design, real-time design and OO design by emphasizing information hiding, module structuring and concurrent task structuring.
Real-Time system	Octopus	Refinement of CODARTS	Based on use cases, static modeling and state charts.
Real-Time system	ROOM	Real-time Object Oriented Modeling	Active objects are modeled using a variation on state charts called ROOMcharts. ROOM model are used as an early prototype of the system.
Large-scale Systems	Use case map	Dynamic modeling of large-scale system	Based on use case concept
Concurrent, real-time and distributed application	COMET (Earlier version)	Collaborative Object Modelling and Design Method	Based on UML 1.3
Large-scale Systems	COMET (Later version)	Collaborative Object Modelling and Design Method	Based on UML 2. Emphasis is on software architecture applications like Object oriented, client/server architecture, component based architecture, service oriented architecture, concurrent and real-time architecture and software product-line architecture

On analysing the above table it can be said that the following areas are using model driven techniques for large-scale software development:

(a) Real-time system (b) Object-oriented system (c) Concurrent system
(d) Distributed system (e) Product-line

In order to make model driven development popular, it requires the following:

a) It must be applied for small-scale software development.
b) Latest approaches like AOP and FOP must be incorporated in modular software development.

As the existing tools and techniques do not cater to the above mentioned areas, hence one can incorporate AOP and FOP in modular software development, particularly in a small-scale software development. This will help to reap the benefits of the latest programming approaches and can also be used by any category of users. Moreover, MDD can become more popular when programs are automatically generated from the existing models. This motivated many researchers to find some efficient mechanisms for both software modelling and automatic code generation with limited success, mostly in highly specialized domains [23].

MDD can reach to a new height only when it can be used by common people for any level of software development. Don Batory's proposal of model driven development as an architectural metaprogramming technique can make MDD usable for common programming level [18]. According to him, MDD is an architectural meta-programming in which models are values and transformations map models to models. This innovative idea can be used for enhancing the capabilities of MDD which can give a new dimension to MDD techniques.

5. Conclusion

Models, modelling and model transformations are the major requirements for developing evolutionary software. Complexity of the software can be reduced by structuring the software through models. Feature-oriented, aspect-oriented programming and model driven development are different types of modular approach. It can be said that the above mentioned areas are common as they all are architectural meta-programming technologies where meta-expressions can be generated. Hence, there is a need to explore the relationship among AOP, FOP and MDD in context of architectural meta-programming. In future, architectural meta-programming can be used for software design and maintenance more efficiently by applying them in sophisticated Integrated Development Environment (IDE) tools.

References

[1] Y. Baldwin and K. B. Clark, "Modularity in the Design of Complex Engineered Systems", In Complex Engineered Systems: Science Meets Technology, Springer-Verlag, pp. 175-205, 2006.
[2] T. Elrad, R. E. Filman and A. Bader, "Aspect-Oriented Programming: Introduction", Communications of the ACM (CACM), vol. 44, no. 10, pp. 29–32, 2001.
[3] S. Herrmann, "Object Teams: Improving Modularity for Crosscutting Collaborations", In Proceedings of International Conference on Objects, Components, Architectures, Services, and Applications for a Networked World (NetObjectDays), vol. 2591, pp. 248-264, 2002.
[4] S. Apel, T. Leich and G. Saake, "Aspect Refinement and Bounded Quantification in Incremental Designs", In Proceedings of Asia-Pacific Software Engineering Conference (APSEC), pp. 796- 804, IEEE Computer Society, 2005.
[5] J. Aldrich, "Open modules: Modular reasoning about advice", In Proceedings of the 18th European Conference on Object-Oriented Programming (ECOOP'05), vol. 3586 of LNCS, pp. 144–168, 2005.
[6] G. Kiczales, "Radical Research in Modularity: Aspect-Oriented Programming and Other Ideas", In Keynote of the International Software Product Line Conference (SPLC), IEEE Computer Society, 2006.
[7] D. S. Dantas and D.Walker, "Harmless Advice", In Proceedings of the International Symposium on Principles of Programming Languages (POPL), pp. 383–396, ACM Press, 2006.
[8] S. Apel, C. K¨astner, T. Leich and G. Saake, "Aspect Refinement", Technical Report 10, Department of Computer Science, University of Magdeburg, Germany, 2006.
[9] G. Kiczales and M. Mezini, "Aspect-oriented programming and modular reasoning", In Proceedings of the 27th International Conference on Software Engineering, New York, USA, ACM Press , pp. 49–58, 2005.
[10] S. Apel, T. Leich and G. Saake, "Aspectual Mixin Layers: Aspects and Features in Concert", In Proceedings of the 28th International Conference on Software Engineering (ICSE), New York, USA, pp. 122-131, 2006.
[11] S. Apel, D. Batory, "When to Use Features and Aspects? A Case Study", In: Proceedings of the International Conference on Generative Programming and Component Engineering (GPCE), pp. 59 – 68, 2006.
[12] D. Batory and S. O'Malley, "The design and implementation of hierarchical software systems with reusable components", ACM Transactions

on Software Engineering and Methodology, vol. 1, no. 4, pp. 355–398, 1992.
[13] Y. Smaragdakis and D. Batory, "Implementing layered designs with mixin-layers", In the Proceedings of ECOOP '98, vol. 1445 of LNCS, pp. 550–570, 1998.
[14] D. Batory, J. N. Sarvela and A. Rauschmayer, "Scaling Step-Wise Refinement", IEEE Transactions on Software Engineering (TSE), vol. 30, no.6, pp. 355-371, 2004.
[15] T.Leich, S.Apel, L.Martinz and G.Saake, "Tool Support for Feature-Oriented Software Development FeatureIDE: An Eclipse Based Approach", In Proceedings *of OOPSLA Eclipse Technology eXchange (ETX) Workshop*, pp. 55–59, 2005.
[16] D .C. Schmidt, "Model-Driven Engineering", IEEE Computer, vol. 39, no. 2, pp. 25-31, 2006.
[17] A. Kleppe, J. Warmer, and W. Bast, "MDA Explained: The Model-Driven Architecture -- Practice and Promise", Addison-Wesley, 2003.
[18] D. Batory, "Program Refactoring, Program Synthesis, and Model-Driven Development", In ETPAS Compiler Construction Conference, vol. 4420 of LNCS, pp. 156-171, Springer, 2007.
[19] H. Masuhara, G. Kiczales and C. Dutchyn, "A Compilation and Optimization Model for Aspect-Oriented Programs", In Proceedings of International Conference on Compiler Construction (CC), pp. 46–60, 2003.
[20] R. Laddad, "AspectJ in Action - Practical Aspect-Oriented Programming", Manning Publications Co., 2003.
[21] G. Georg, I. Ray and R.France, "Using Aspects to Design a Secure System", In Proceedings of the Eighth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'02), pp. 117-126, 2002.
[22] H. Gomaa, "Software Modeling and Design", Cambridge University Press, 2011.
[23] B. Sellic, "The Pragmatics of Model-Driven Development", IEEE Software, vol.20, no. 5, pp. 19-25, 2003.

Mahua Banerjee pursued her B.Sc. (Physics Hons.) from Calcutta University, PG Diploma in Computer Application and Masters in Computer Application in the year 1984, 1990 and 2004 respectively. She received her Ph.D. degree in Computer Science from Indian School of Mines, Dhanbad, India in 2014. She is teaching since 1990 and currently she is an Asst. Professor in Dept. of Information Technology in Xavier Institute of Social Service, Ranchi. Her current research interests include Software Engineering, Modular Programming and e-Commerce.

Sushil Ranjan Roy was awarded the B.Sc. Chemistry honours degree from Ranchi University, Ranchi in the year 1982. He was awarded the ICAR fellowship in 1992. He worked as a lecturer in Chemistry from 1984 to 1987. In 1987 he joined Xavier Institute of Social Service, Ranchi as a lecturer in Computer Science, where he is now working in the capacity of Associate Professor. He has been teaching at the Post Graduate level for the last twenty seven years. In the year 1990 he was awarded the prestigious ECPR scholarship, which funded his stay and course fee at the Summer School at Essex University, England. His research interests are in the field of Software Engineering, ICT for development and Mathematical Computing.

Satya Narayan Singh received B.Sc. and M.Sc. degree in Mathematics from Ranchi University in the year 1987 and 1993 respectively. He received the Ph.D. degree from Ranchi University in the year 2010. He is currently Professor and Head, Department of Information Technology of Xavier Institute of Social Service, Ranchi. His current research interests are in Hadamard matrices, e-Governance and Software Engineering.