CHAPTER 2

LITERATURE SURVEY

This chapter describes the survey of existing literature on multiple views. Later, it presents literature survey conducted on frameworks for tool comparison and stakeholder assessment. It also provides a brief introduction to the recommendation systems for software engineering and existing recommendation systems.

2.1 MODEL-DRIVEN SOFTWARE EVOLUTION

Model Driven Engineering (MDE) brings advantages like reducing development and maintenance effort by considering models as first class citizens. Thus models have become integral part of the software system and evolution process. MDE is unification of initiatives that aim to improve software development by employing high-level, domain specific models in implementation, integration, maintenance and testing of software systems [13]. It also requires a new style of evolution. In traditional software evolution, the platform is fixed. Migration to a new platform is a tedious event. Thus, Model-Driven Software Evolution (MoDSE) is essential. For realizing MoDSE there is a need for investigating and integrating the basic techniques like construction of model development environment. Models are developed and transformed to code by using modeling language. For legacy systems there is a need to transform code to model. Hence, MoDSE requires many types of transformations such as from model to code, from code to model, and model to model [1]. Evaluation of the methods and techniques used to perform transformations in MoDSE is also required. The key innovation that model-driven software evolution requires is a unification of several activities like reverse engineering, program transformation, code generation and model transformation.
Thus, this thesis aims to focus to view above mentioned activities in various aspects (e.g., stakeholder concerns). MDE handles these activities in various domains like specific software applications. So, MDE deals with the domain specific languages.

2.1.1 Domain Specific Modeling

MDE is a promising approach to address platform complexity by combining Domain Specific Languages and Model Transformations [29]. MDE is related to the field of Domain-Specific Languages which fills the gap between general purpose languages and particular application domains, by providing a language with notation and concepts geared to the domain. The need for Domain-Specific Languages (DSL) is clearly demonstrated by their presence in domains such as language recognition (YACC, ANTLR, SDF), graphics (SVG, VRML), querying (XQuery, SQL, XPath), text processing (Perl, Sed), document transformation (XSLT), mathematics (Mathematica, Matlab, Fortran, Fortress, Magma) and enterprise query languages (EJBQL for Enterprise JavaBeans, JDOQL for Java Data Objects, ODMG’s OQL, Hibernate’s HQL, JMS message selectors) etc [1].

Domain-Specific Modeling (DSM) raises the level of abstraction beyond programming by specifying the solution directly using domain concepts. The final products are generated from these high-level specifications. It is automatically possible because both the language and generators need fit into the requirements of only one company and domain. Models can be attractive because they allow a problem to be precisely described in a way that avoids examining into technological detail. As technology becomes more and more complex, modeling is increasingly necessary in order to be productive. Another advantage of models is that they allow the problem to be described using terms and concepts that are familiar to people who work in the domain of the problem.
Domain-Specific Languages (DSL) can be created for numerous problem domains like telecommunications, investment banking, public transport, space exploration, and in many other areas. There is a need for a process by which a model of the domain can be analyzed and validated, as well as transformed, often through several steps, into deployed and executing software. This process involves analysis, development, and validation of models in several different domains. To perform these activities tools and processes are required to transform between one kind of model and another. Traditionally, software development has been a series of mappings from the domain idea, to design models, and on to source code. These mappings tend to be slow and lead to errors and duplication of effort in problem solving, design, and coding [28].

DSM addresses these problems by removing the resource-intensive and error-prone mappings, aiming to solve the problem only at domain level. This raises the level of abstraction of models, reduces the amount of information that needs to be modeled, reduces mapping by moving the modeling language closer to the domain as perceived by designers, and improves the quality and scope of code generators. A domain is thus often as small as a given range of products of a given company. The more specific the domain can be, the higher the productivity benefits of applying DSM [82]. Typically, MDE approaches are based on modeling languages that offer abstractions focused on a particular domain. Such languages are referred to as domain specific modeling languages (DSMLs). Recently much attention is on development of DSMLs. These languages can range from highly individual domain such as ‘rail road planning’ application to broader domain like ‘embedded systems’ [86]. B.Graaf and A.Van Deursen proposed a Model-Driven Architectural Views (MDAV) framework which describes the advantages of DSMLs [20]. In MADV, connection between views and models are made through UML diagrams. So, these views became model driven to visualize DSML. Views in MDAV are Module-Uses view, Component-Connector View in which Metamodel, UML Mapping, Model creation and documentation are developed and visualized. Approach for MDAV framework is evaluated in the domain of software architecture. Instead of focusing on specific
domain the main aim in this thesis is to concentrate on various activities like model transformation, model mapping, model integration, model creation, which leads to evolution of models in the context of MoDSE. These activities are viewed by different stakeholders in different views. For this purpose multiple views are proposed in the thesis.

2.2 VIEWS

In Model-Driven Software Evolution understanding evolution of models is essential in stakeholder’s aspect. Views play an important role in understanding different models in various perspectives. Views can capture the information about models during evolution. ChristainF.J.Lange, et.al, proposed a framework consisting of UML model elements, their properties, and software engineering tasks that form a basis to develop new views of UML models and related information [23, 24]. Based on this framework they proposed eight views to support different tasks. These views are UML based views, which maintains model evolution and quality. iACMTool is a prototype tool to tackle impact analysis and change management of analysis/design documents in context of UML based development [15]. This taxonomy consist views such as static (class diagram) view, interaction (sequence diagram) view and state chart diagram view. These views support UML models.

Multi View Software Evolution (MVSE) is a UML based framework for Object-Oriented software [81]. In MVSE, evolution of complex systems is a process in which transformations are successively applied to multiple views of software (represented by models), until objective criteria are satisfied. A view reflects the perspective of a stakeholder on a system’s application and behavior. In MVSE stakeholders initiate changes to systems and describe these changes in the context of stakeholder views. Rene Keller et.al, introduces concepts of multiple viewpoints and multiple views in engineering change management [78]. Change Prediction Method (CPM) tool implements the change prediction. Stephen Cook et.al, proposed an approach to understand software evolution
This approach looks at software evolution from two different points of view. One is dynamic view point, which investigates software evolution trends in models and second is static view point which studies the characteristics of software artifacts to see what makes a software system more evolvable. In this thesis concept of view points, views, concerns, stakeholders and their roles are considered from IEEE 1471 standard which serves as background for the proposed multiple views [39]. The above mentioned frameworks describe the multiple views and viewpoints for traditional software evolution and change management, in which UML models are considered. Hence, there are no such views and viewpoints exist in the literature, to address the stakeholders concerns during evolution of models in MoDSE. Proposed views are validated analytically against Lehman’s laws of software evolution.

2.3 SOFTWARE EVOLUTION AND VISUALIZATION

Visualization aims at enhancing information understanding by reducing cognitive overload. Using visualization methodologies and tools, people are often able to understand the information presented in a shorter span of time and to a greater depth. Visualizing can also refer to the process of determining mappings between abstract or real-world objects and their graphical representation. Thus, by considering visualization concept in MoDSE, evolution of models can be visualized effectively which in turn helps stakeholders to understand evolution process. Therefore, major contribution of this thesis is proposing a framework for visualizing model-driven software evolution. In this thesis the term “visualization” refers as a process of mapping the evolution of models to stakeholder concerns. Several frameworks found in the literature which were proposed for areas like software evolution, software visualization, model driven software visualization and some of them are discussed here. Source Viewer 3D (sv3D) is a Software Visualization framework that builds on the SeeSoft metaphor[45]. sv3D can show large amounts of source code in one view. Object based manipulation methods and simultaneous alternative mappings are available to the user. The types of
user tasks and interactions that are supported by sv3D, is not directly related to solving/visualizing specific software engineering tasks and it is a prerequisite for a software visualization tool.

Architecture to Support Model Driven Software Visualization [3], borrows the field of Model Driven Engineering (MDE) to assist with the creation of highly customizable interfaces for Software Visualization. In order to validate the architecture, MDV framework for Eclipse was developed. Model Driven Visualization (MDV) is intended to address the customization of information visualization tools, especially in program comprehension domain. MDV architecture describes how to leverage the work done in Model Driven Engineering community and apply it to the problem of designing visualizations tools. Graphical Modeling Framework (GMF) project for Eclipse has facilities to allow modelers to define graphical editors for their data [23]. These graphical editors can be used as viewers, however, the views they support are limited to simple graphs with containers. GMF project currently lacks the ability to specify “Query result” visualizations. An Open Framework for visual mining of CVS based software repositories has three major aspects such as data extraction, analysis and visualization [14]. An approach was proposed for CVS data extraction and analysis. CVS data acquisition mediator used to extract the data from CVS repositories. Analysis techniques are used to analyze the raw data retrieved from the CVS repositories from CVS Querying. It also provides the comparison of the open source projects. CVSgraph is a software tool used to visualizing project at file level. This open framework does not provide the visualization of models, it provides for program at file level only. CVSscan is a tool in which a new approach for visualization of software evolution was developed [15]. The main audience targeted here is software maintenance community and the goal is to provide support for program and process understanding. This approach uses multiple correlated views on the evolution of a software project. The overall evolution of code structure, semantics, and attributes are integrated into an orchestrated environment to offer detail-on-demand. It also provides the code text display that gives a detailed view on both the composition of a fragment of code and its evolution in time. It is focused on the evolution of individual files.
Above discussed frameworks have contributed to the groundwork needed to derive and construct framework for visualizing Model Driven Software Evolution. However, none of these frameworks provides a solution. In order to visualize and understand evolution of models in model driven evolution process, stakeholders need multiple views and tools to satisfy their concerns. By applying proposed framework, stakeholders can determine the tools that satisfy their concerns and comparison of such tools is also possible. To determine which tools are essential to understand various concerns of stakeholders in MoDSE, literature survey has gathered various existing strategies for comparing CASE tools which are presented in the next section.

2.4 CASE TOOLS

A widely accepted statement is that tools are needed in order to put in practice in an effective way all the techniques involved in the construction of a Model Driven Software Development (MDSD) based method. Without tools which automate the steps that must be carried out during the application of such kind of methods, most of the promised benefits cannot be obtained [22]. This statement reveals the vital role of the automated tools in any software engineering task. So, this thesis aims to collect information about the various tools. These tools are considered for selection, assessment and comparison under a uniform platform. To have a comparison of various tools a framework and recommendation system is proposed for MoDSE. Only MDA and UML tools are considered in this research because of suitability of these tools in MoDSE. This kind of comparisons exists in literature and few of them are discussed here.

Integrated constrain support in MDA tools evaluated in [47]. Different tools were classified in categories like CASE tools, MDA specific tools, MDD methods and OCL tools. MDA tools considered in the classification are closer to MDA standard. Only few tools such as Poseidon, Rational Rose, Magic Draw, Objecteering/UML, Together, ArcStyler, OptimalJ and AndroMDA etc., have selected for the comparison and evaluation purpose. Support of current tools regarding an automatic generation of the code required to enforce integrated constraints specified in a Platform Independent Model
(PIM) also surveyed. The main shortcomings encountered are lack of expressivity and efficiency in integrated constraints.

A short comparison of three MDA tools was presented in [73]. It was focused on concepts behind the tools as well as how to use them. The tools considered for comparison are ArchitectureWare, AndroMDA, and openMDX. The comparison strategy presented might be useful for evaluation purposes to know the differences in approach, features and concepts which are to be considered as the implementation of OMG’s MDA specification. Computer Aided Software Engineering (CASE) tool community is an open access web application [111]. Different categories of CASE tools like MDA, UML, reverse engineering, agile modeling etc., are listed in this community. Key functions and external links of each tool are provided. Connecting to other forums and groups is also possible. Users allowed to upload new CASE tools and can share their ideas. It also provides the external link to various articles about the case tools. Users have a freedom to vote for the best tool. ‘modelbased.net’ forum is dedicated to tools and information related to model-driven system development, aiming at supporting OMG’s vision MDA are provided [112]. This website provides overview and resource links of MDA oriented tools, UML, MOF, and Model Transformation tools.

MDA tools are categorized in three ways [62]. First, whether the tool is open source or commercial which will help to choose a tool that is right for the culture of an organization, among other things. Second, whether the tool offers a partial or complete MDA solution that helps in such considerations as cost, quality, and flexibility. The final category verifies automatic code generation from model or execution of model. It is not mentioned what are the tools that fits into specified categories. IBM Rational software has several products that support MDA and Model Driven Development (MDD) in varying capacities [78]. These tools fall into three basic categories such as general-purpose, domain-specific, and supporting. For example, Rational Software Architect is in general-purpose category, IBM Rational Systems Developer in domain-specific category, and IBM WebSphere Business Modeler in the supporting category. The usage of these tools in MDA was
described. Only IBM products are considered for categorization. Above mentioned forum, panel and community are only the information content of the tools which do not provide any recommendations and/or suggestions about tools for different activities in the context of MoDSE. Therefore, this research mainly aims to propose a recommendation system. Survey on recommendation systems for software engineering is discussed in next section.

2.5 RECOMMENDATION SYSTEM CONCEPTS

General definition and description proposed by the organizers of the ACM International Conference on Recommender Systems [57]. Recommendation systems are software applications that aim to support users in their decision making while interacting with large information spaces. They recommend items of interest to users based on preferences they have expressed, either explicitly or implicitly. Ever-expanding volume and increasing complexity of information has therefore made such systems essential tools for users in a variety of information seeking activities. Recommendation systems help to overcome the information overload problem by exposing users to most interesting items, and by offering novelty, surprise, and relevance [57]. Recommendation Systems for Software Engineering (RSSE) match this definition in their aim to support developers in decision making. Martin P. Robillard et al., presented the basics such as what is RSSE, what they do for developers, and design dimensions etc., [57]. RSSE might need to provide or infer the user’s characteristics, kind of task being conducted, task’s specific characteristics and user’s past actions or those of user’s peers. Considering some of the particulars of software engineering RSSE is defined as “It is a software application that provides information items estimated to be valuable for a software engineering task in a given context”. The diversity makes generalizations about RSSE architecture difficult, but most involve at least three main functionalities [57]:

- A data collection mechanism to collect the data and artifacts in a data model.
• A recommendation engine to analyze the data model and generate recommendations.
• A user interface to trigger the recommendation cycle and present its results.

Three major design dimensions for RSSE considered as follows:

• **Nature of the context:** The recommendation context is a core RSSE concept. It can be explicit, implicit, or a hybrid of these strategies. User-interface interactions such as entering text, selecting elements, or dragging and dropping elements are explicit context. User’s interest, track and react to developer’s actions etc., are implicit context.

• **Recommendation Engine:** RSSEs must analyze more than context data to make their recommendations. Additional data can include the project’s source code, history of system change, emails posted to mailing list and bug.

• **Output Modes:** Most existing RSSEs operate in pull mode and produce recommendations after a developer’s explicit requests. Some RSSEs operate in push mode, delivering results continuously, for example ‘Dhruv’[7].

RSSE can cross design dimensions. For example, recommendation engine can take developer’s interactions into account, allowing developer to flag bad recommendations to eliminate them from future results. Ranking mechanism can be locally adjustable. But existing RSSEs are often limited in the ranking mechanisms they offer. RSSEs evolve not only with developers’ needs but also with the nature of available data and development of technologies. To date, most RSSEs have focused on recommendations related to software development artifacts, particularly source code. RSSEs typically recommend code-to look at, change, or reuse. However, recommendations could address many other aspects of software development. For example, recommendations for quality measures, tools, project management, and people could support an ever-widening array of software
engineering tasks. Therefore, this thesis proposed recommendation system which considers MDA and UML tools suitable for MoDSE. The next section presents literature survey on existing RSSEs.

2.6 RECOMMENDATION SYSTEMS FOR SOFTWARE ENGINEERING

Many RSSEs are available in literature but widely used and relevant recommendation systems are discussed. Strathcona system retrieves relevant source code examples to help developers use frameworks effectively. For example, a developer who’s trying to figure out how to change the status bar in Eclipse IDE can highlight the partially complete code (the context) and ask Strathcona for similar examples. Strathcona extracts a set of structural facts from the code fragment. Strathcona uses PostgreSQL queries to search for occurrences of each fact in a code repository. Next, it uses a set of heuristics to decide on the best examples, which it orders according to how many heuristics select them. It returns the top 10 examples, displaying them in two formats - a structural overview diagram and highlighted source code. Developers can also view a rationale for a proposed example and prototype is also available. Dhruv is a Recommendation System which recommends people and artifacts relevant to a bug report. It operates chiefly in open source community, which interacts heavily via the Web. Using a three-layer model of community (developers, users, and contributors), content (code, bug reports, and forum messages), and interactions between these. Recommendations are generated according to the similarity between a bug report and terms contained in the object and its metadata. Finding the right software experts to consult can be difficult, especially when they’re geographically distributed.

Expertise Browser is a tool that recommends people by detecting past changes to a given code location or document. Tool described in this paper solves an expertise finding problem in a way that meets the proposed way of identifying project related expertise. One can easily find people who worked on particular parts of a project, and can see easily how their experience is distributed over
the product, i.e., highly specialized or broad, and exactly where the contributions occurred. Appropriate filtering of expertise atoms allows display of tool, language, or release and other types of expertise. Second, this is accomplished by using data that are collected automatically. It imposes no burden on individuals to describe their expertise. It also provides alternatives to querying project architects and other well known experts, increasing chances that the user will find an expert quickly, and reducing overload on “expertise experts.” ParseWeb recommends sequences of method calls starting from an available object type and producing a desired object type [100]. ParseWeb analyzes example code found on the Web to identify frequently occurring call patterns that link available object types with desired object types. Developers use this tool by specifying available and desired object types and requesting recommendations. Recommendation system is a simple software application which generates recommendations and provides decision technologies which can be exploited for proactively supporting stakeholders in their decision making [94]. Recommendation system might use activity logs to deduce questions developers ask, and then coach them automatically on appropriate, possibly unfamiliar tools or features to answer those questions more efficiently [32]. Thus, this thesis proposed a recommendation system which generates recommendations for tool selection in the context of MoDSE and it is different from forums and communities.