HAL
archives-ouvertes.fr

Modeling Practices in Open Source Software

Omar Badreddin, Timothy Lethbridge, Maged Elassar

▶ **To cite this version:**

Omar Badreddin, Timothy Lethbridge, Maged Elassar. Modeling Practices in Open Source Software. 9th Open Source Software (OSS), Jun 2013, Koper-Capodistria, Slovenia. pp.127-139, 10.1007/978-3-642-38928-3_9 . hal-01467586

HAL Id: hal-01467586
https://hal.inria.fr/hal-01467586

Submitted on 14 Feb 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution| 4.0 International License

Modeling Practices in Open Source Software

Omar Badreddin[1], Timothy C. Lethbridge[1], Maged Elassar[2]

[1]University of Ottawa
800 King Edward

[2]IBM Ottawa Laboratories

770 Palladium Dr. Ottawa, Ontario, Canada
Ottawa, Ontario, Canada
obadr024@uottawa.ca, tcl@site.uottawa.ca, melaasar@ca.ibm.com

Abstract. It is widely accepted that modeling in software engineering increases productivity and results in better code quality. Yet, modeling adoption remains low. The open source community, in particular, remains almost entirely code centric. In this paper, we explore the reasons behind such limited adoption of modeling practices among open source developers. We highlight characteristics of modeling tools that would encourage their adoption. We propose Umple as a solution where both modeling and coding elements are treated uniformly. In this approach, models can be manipulated textually and code can be edited visually. We also report on the Umple compiler itself as a case study of an open source project where contributors, using the above approach, have and continue to routinely commit code and model over a number of years.

Keywords: Open source, UML, Modeling, Coding**,** Version Control, Model Merging. Model versioning, Umple, Model Driven Architecture.

1 Introduction

Open-source software (OSS) is witnessing increasing adoption and impact [1]. OSS development communities are collaborative, typically geographically distributed, working together to build software, often over multiple years with hundreds or thousands of contributors. OSS teams share common characteristics. For example, they use open-source version control systems, wikis, issue tracking systems, and automated test suites. Many teams use continuous integration – automated build servers to integrate contributions frequently and generate updated systems within minutes after a new commit.

However, code remains a key development artifact. Documentation and especially models are typically either absent or nearly so. This is despite evidence that model-driven development can increase quality and productivity [2]. In this paper, we propose an approach to help bring modeling to open-source development without disrupting the culture and methods open-source developers have found successful.

The use of modeling (e.g., UML) is considered to be a good practice that is expected to positively affect the quality of software [2]. A recent survey of modeling practices in the industry reveals different levels of modeling adoption [3]. They range from full adoption of model-driven engineering, where engineers rely exclusively on models to generate complete running systems, to using models for documentation only. The survey also reveals that modeling adoption in general remains much lower than what would be desirable.

By only using a set of simple and readily-available tools, OSS communities are able to foster collaboration among a large number of members who often do not know each other. Despite this success, these teams rarely, if ever, use modeling as part of their development activities.

We surveyed the 20 most active open source projects according to Ohloh [4] based on the number of commits. The top of the list is Chromium [5], with about 7 million lines of code, 150,000 commits, and more than 1,300 contributors. Chromium is mostly written in C and C++ (72%). When summing up all hours spent on the development, Chromium consumed 2,000 years of effort. The rest of the list includes projects such as Mozilla Core, WebKit, and GNOME.

The majority of these projects are written in C or C++ (87%). A small percentage of commits (0.3%) were XML based, which could indicate some use of modeling. However, none of these 20 projects listed any modeling notation as a notation where commits were accepted. We can therefore conclude based on this survey that there is no evidence that contributors commit models in any significant numbers. This finding validates our earlier survey results [3] and is in line with other existing observations [6, 7].

Reasons behind this low adoption in the industry in general, and in open source projects in particular, have been investigated in our previous work [3]. In summary, low adoption can be attributed to the following reasons:

- Code generation doesn't work as well as it needs to;
- Modeling tools are too difficult to use;
- A culture of coding prevails and is hard to overcome;
- There is a lack of understanding of modeling notations and technologies;
- The code-centric approach works well enough, such that the return on investment of changing is marginal, yet the risks are high.

The focus of this paper is proposing a new development paradigm that provides tight integration of modeling and coding styles. We demonstrate that this paradigm has the potential of increasing modeling adoption by presenting a model-driven programming environment called Umple, which we developed as an open source project.

2 Proposed solution

Umple is a fully functional and executable language with a textual syntax that allows uniform code and model development. Umple allows model abstractions from UML to be embedded into code written in Java, PHP and other programming languages.

This raises the abstraction level of modern object oriented programming languages to be on par with modeling artifacts. In the remainder of this paper we focus on the Java context.

Umple code can essentially be straight Java code, if desired, but most commonly, it consists of a synergistic mix of Java code, UML associations, and UML state machines all in the same textual format. It is therefore very easy for an open source team to move from Java to Umple. If they do so, their code base tends to shrink considerably as much of their 'boilerplate' Java code can be abstracted out and replaced by more concise modeling constructs.

Umple's compiler and code generator are written in Umple itself. Umple also has a web-based modeling environment that does not require any tool installation.

Umple was developed to help enhance modeling adoption by software engineers in general. It started as a closed source project but was then released as an open source project in 2010. Since then, Umple has had more than 30 contributors who routinely contribute code with modeling elements directly embedded. Umple's integration server runs a suite of more than 2,700 test cases to ensure that the model and code are consistent and produce a working system.

3 Introduction to Umple

A key philosophy behind Umple is that modeling and coding are just different perspectives of the same development practice, with the difference being a question of abstraction. Modeling constructs such as UML associations and state machines are more abstract than traditional code elements like while loops, yet both appear together in Umple code. Complex Umple modeling constructs are typically more readily understood when rendered visually as UML diagrams, while traditional coding constructs are nowadays more commonly best understood when shown as indented text, but it need not be this way only. The Umple environment enables users to model textually and visually (with updates to one view applied instantly to the other view).

Consider the Umple sample code in Listing 1.

```
1    class Person { }
2
3    class Student {
4      isA Person;
5      Integer stNum;
6      status {
7        Applied {
8          quit -> Quit;
9          enroll [!hold] -> Enrolled;
10       }
11       Enrolled {
12        quit -> Quit;
13        graduate-> Graduated;
14       }
15       Graduated {}
16       Quit {}
```

```
17      }
18      * -- 0..1 Supervisor;
19    }
20
21    class Supervisor {
22      isA Person;
23    }
```

Listing 1: Sample Umple code/model [8]

The structure of the code is familiar to a Java programmer. However, many of the lines of this example represent UML elements embedded textually in the code. Line 18 is a UML association, and indicates that class Student can have an optional Supervisor, while a Supervisor can have many Students. Lines 4 and 22 are Umple's way of representing generalization (i.e. designating a superclass). Umple Online [9], the Umple web-based environment, would render the above code as a UML class diagram, as shown in Figure 1.

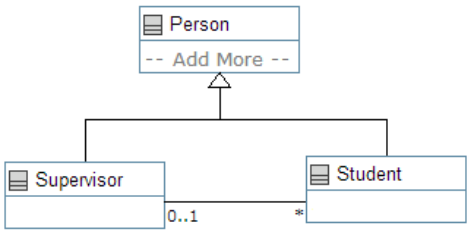

Figure 1: A Class Diagram in UML

Lines 6 to 17 describe the behavior of the Student class as a state machine, which can be depicted as in Figure 2. When a Student object is created, it goes initially to the Applied state. Depending on the events that occur next, and whether or not the 'hold' boolean variable is true, the Student object moves from one state to another state.

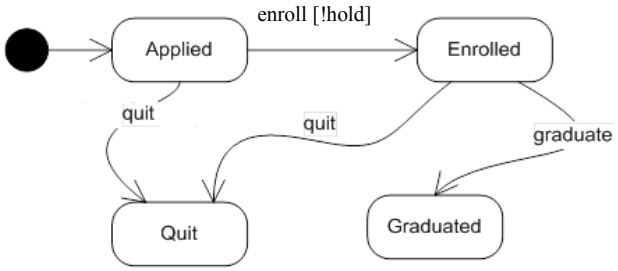

Figure 2: State Diagram

Umple is not intended to replace the need for visual models. As illustrated in the following sections, the Umple compiler has both a textual and a visual editor. Both views are synchronized automatically, since a change in the model will always have an equivalent change in the textual representation. Of course, not all code changes

have an impact on the visual representation, since not all aspects of the code are visualized in every diagram type.

3.1 Code generation from Umple

The Umple philosophy is to combine code and model into one type of artifact. Hence, Umple does not have the typical process of code generation (model to editable code) nor reverse engineering (edited code to model) found in other modeling tools. An Umple program, consisting of model and code, is compiled into an executable system directly. It is not necessary to reverse engineer Umple code to create a model, since it already is a model.

3.2 Language independent modeling

The main strength of UML as a modeling language is being platform and implementation independent. This allows the model in Figure 2, for example, to be used to generate code in Java, C++, or any other programming language. This independence is also maintained in Umple. Users of Umple can chose to write a system in any language of their choice as long as its compiler is available. Umple currently provides full support for Java, PhP and Ruby, with growing support for C++. Umple can also generate XMI and Ecore [10] representations for integration with existing modeling tools.

4 Case Study

In this section, we assess the effectiveness of using model-driven programming languages, such as Umple, to encourage adoption of modeling in open-source projects.

Our case study is on Umple itself as an open source project, hosted in the Google Code repository [11]. The source code of the Umple compiler is written in Umple with Java as the target language and the language of the embedded methods. Contributors need to learn the syntax of Umple before contributing. But since the syntax is very similar to Java and resembles what would appear in a UML diagram, learning Umple does not add significantly to the time needed for contribution.

As with a typical open source project, there is a little up-front effort required to set up the development environment, which includes downloading the code, installing tools such as Eclipse (with the Umple and Jet plugins) and Ant, getting the appropriate project permissions, etc.

Umple was released as open source software in 2010. However, all of the source (model/code) artifacts was stored in a Subversion repository for over 5 years; it is possible therefore to analyse the source changes, including the UML underpinnings of the Umple compiler. Because the model is represented textually in Umple, there is no additional infrastructure investment for model versioning and comparing.

4.1 Objectives and Research Methodology

The objective of this study is to investigate the applicability of model oriented programming in open source projects. This investigation is carried out by analyzing how model orientation was used in developing the Umple platform.

In previous work, we have provided empirical evidence that model oriented code has significant comprehension value over pure Java code [12]. The empirical evidence is a result of a controlled experiment comparing model-oriented code, traditional object-oriented code, and UML models. The experiment included both student and professional participants and proceeded by posing questions on a set of different modeling and coding artifacts. The question and answer sessions were recorded and analyzed to infer comprehensibility of the different code and modeling snippets. The experiment provides evidence that suggests that model oriented code is at least as comprehensible as UML visual models. Therefore, in this paper, we limit our focus to investigation of the applicability of using model orientation in open source environments.

The design of Umple was carried out over more than 5 years. The technology is still under extensive development and iterative improvements. Design of Umple was influenced by the following studies.

- Study of early adopters [13]. This is a study that was conducted during the early design phases where early adopters were interviewed and filled up questionnaire that reflects their attitudes towards model oriented code.
- Experiences of applying Umple in industrial development projects [14]. By using the technology in industrial projects, we were able to refine the design and implementation of the platform. In particular, we improved scalability aspects of the technology.
- User evaluations and experimentation [15]. Such studies provided empirical evidence to support our claims about the comprehension value of model oriented programming technology. These studies have also given us insights on how users actually go about exploring the usage of such technology.

Design is both an art and a science, making it challenging to satisfactorily evaluate design work. Recently, research on design has witnessed a significant uptake in several domains. This increased uptake has encouraged researchers to define a systematic methodology of evaluation design [16]. In summary, this evaluation methodology relies on iterative design evaluation cycles with key stakeholders. This methodology also adopts empirical evaluation of design by measuring to what extent a design satisfies its intended goals.

We adopted the same design evaluation methodology mentioned above in the design of Umple. We implemented a prototype of Umple that embodied many of our design objectives. We used the prototype to iteratively assess to what extent the design met its objectives and we collected feedback and evaluation from users. This case study does not discuss the intermediate designs, but rather focuses on the final design after completing the process of iterative refinement.

4.2 Development environment

Developers of Umple use a variety of development environments. For simple tasks, the Umple web-based editor [9] can be used to easily write code and visualize the equivalent UML model. Figure 3 is a snapshot of the UmpleOnline system. On the left side there is the code/model textual editor. On the right side, there is the corresponding visualization editor for the model. Edits on either side are automatically reflected on the other side. If a user modifies the default layout of the visual rendering of the model, the new layout information is stored textually. This is to support preserving the layout when a user returns to the same visualization again. This layout information is committed and versioned in the repository.

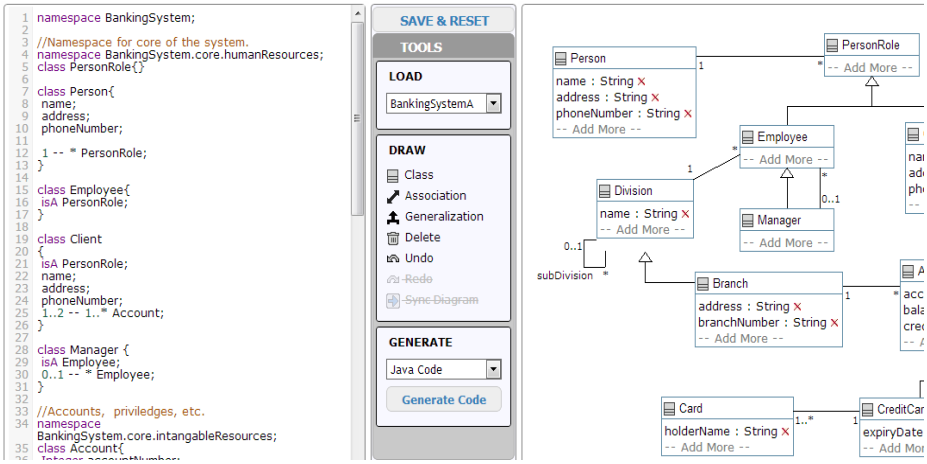

Figure 3: Umple Online

Umple Online is suitable for a quick demonstration of concepts and for relatively simple development tasks. However, for larger and more elaborate tasks, the Eclipse based plug-in is preferred. The added functionality provided by the plugin includes code highlighting, context assist, outline view, error and problem reporting views, as well as compiling. Finally there is a command line version of the compiler, which is used for automatic building, and is therefore preferred by some developers.

4.3 Developer Contributions

Contributions to Umple are represented in text, as in other open source projects. The key and only difference is that Umple text represents both model and code elements. Once the Umple continuous integration server senses a commit, it automatically triggers a build. The build process first compiles all Umple source files to generate a new executable Umple compiler. Then, the build process runs Umple test cases to verify that the latest commit did not break existing functionality. The new compiler then builds itself, as yet another large test case. If the tests are all successful, the new build is completed and deployed.

4.4 Umple source versioning

Umple developers review code and model changes in the same way. Since model and code in Umple are textual; changes can be reviewed using the repository diff functionality (Figure 4).

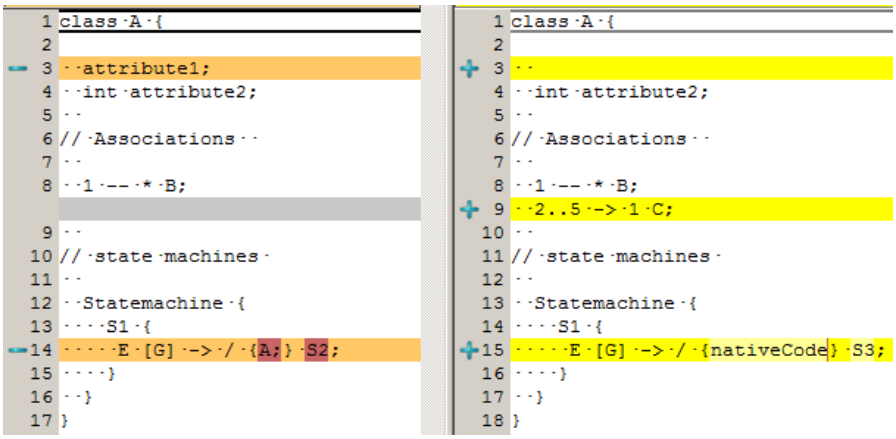

Figure 4: Code and model revisions

4.5 Quantitative Assessment of Model and Java Code in the Umple Code Base

In this section, we study what portion of our case study Umple code base consists of 'model' code (i.e. Umple's textual rendering of UML), and what portion consists of standard Java code . We also present a brief analysis of how this varies over time.

Our investigation is based on analyzing the revision history of three artifacts (the Umple associations and attributes portion of the metamodel, the state-machine portion of the metamodel, and the parser) as shown in Table 1. The period covered is from October 2008 to February 2012. We chose these artifacts because they are frequently updated, have existed for a long time, belong 'together' in the architecture (the parser populates the Metamodel), and have different distributions of model vs. Java (the metamodel files are mostly model, and the parser is mostly Java). We chose a subset of Umple to simplify the data, and we chose three components to see if the results are consistent between them. This is an exploratory case study only, so we have not attempted to analyse the entire system.

The Umple metamodel is the schema written in Umple of all Umple elements in the system being compiled and is used to define different modeling and code elements within the Umple language. Changes to the metamodel occur when a new Umple language element is defined or when a change is done to an existing element.

The number of lines of code for the associations and attributes portion of the metamodel is 1623, 19% of which are modeling elements and the rest are Java code.

The second artifact is the state machine portion of the metamodel, which is the schema of the state machine related elements. The artifact has 315 lines, 24% of which are model.

The third artifact is the parser, which is a component of the Umple compiler. The parser contains mostly Java code (690 lines), 4% of which is model.

Table 1: Model and code contributions

File	changes	% Model only	% Java only	% Model and Java change	Number of lines (Feb. 2012)		
					Total	Model only (%)	Java only (%)
Attributes and associations metamodel	1414	15%	40%	45%	1623	307 (19%)	1316 (81%)
State machine metamodel	779	18%	25%	57%	315	77 (24%)	238 (76%)
Parser	1242	1%	97%	2%	690	26 (4%)	664 (96%)
Average						**15.6%**	**84.3%**

Comparison of Model vs. Java code in terms of the volume of their change relative to their size: The number of changes and the percentages affecting model only, Java only or both are given in the first four columns of Table 1. It would typically be challenging to produce these numbers automatically since Umple treats both model and code uniformly. Hence, in other systems one might have to manually inspect the changed lines or write a special-purpose parser to judge whether they are code or model related. However, we took advantage of the Umple project's coding guidelines, which advise developers to keep modeling elements at the top of the file or in separately-included files whenever possible, to quickly classify the majority of changes. We also used Fisheye [17], a repository visualization tool to help verify our classification estimates. Finally, we verified our numbers by asking three experienced Umple developers to produce them on their own. No significant discrepancies were evident.

Since the Umple metamodel includes a much higher portion of modeling elements than the parser, while the parser includes a much higher portion of Java code than the metamodel, we expected that the changes to those artifacts follow similar proportions (more model changes in the metamodel; more java changes in the parser). Table 1 show that this pattern was indeed observed. In fact, it was valid across different developers. In other words, the change patterns of any particular developer are similar to what is shown in Table 1.

Comparison of Model vs. Java changes over time: Umple was initially written purely in Java. Incrementally, more modeling abstractions were added to the language, starting with attributes, then associations, and finally state machines. The more modeling support got added, the more we were able to refactor the code base to

take advantage of the modeling abstractions. One would therefore expect to find an upward trend in the percentage of modeling changes and that this trend would persist over time. To our surprise, we noticed a different pattern: modeling changes exhibited a downward trend for most of the time period of our observations (Figure 5). Our explanation is as follows: As the code base was refactored to take advantage of more modeling abstractions, modeling changes were initially extensive as compared to Java changes. But as the refactoring slowed down, more changes occurred to the Java elements to add features, largely algorithms and computations that typically do not involve creating new or changing existing modeling elements.

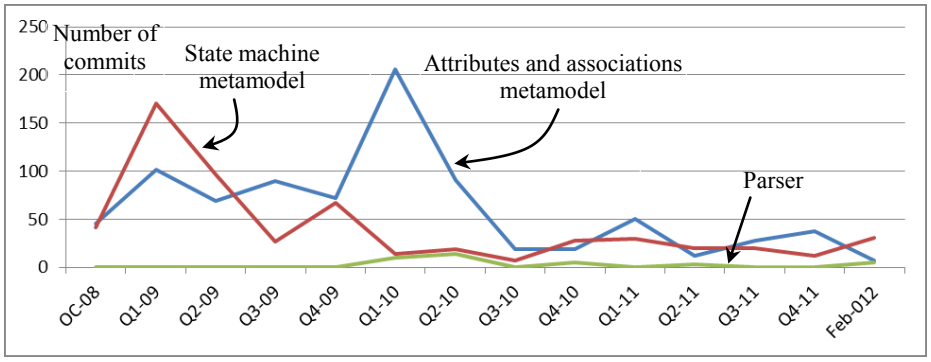

Figure 5: Model changes trend over time

Figure 5 illustrates the analysis for the same three artifacts; the state machine metamodel, the attributes and association metamodel, and the parser. The figure aggregates model only changes with code and model changes, starting from October 2008 and until February 2012. The data is aggregated quarterly.

The core attributes and association metamodel was developed prior to October 2008. The spike in change activity to Attributes and association metamodel (around Q1 2010) corresponds to refactoring of the metamodel that took place as the responsible researcher was transitioning out. The state machine metamodel was developed during 2009. There was little refactoring performed on this metamodel since it was developed after many modeling abstractions were already supported by Umple. In other words, the state machine metamodel was written in a version of Umple that supported attributes and associations.

The downward trend is more evident in the case of the state machine metamodel than in the attributes and association metamodel. The number of model lines in the parser is minimal; hence, one cannot elicit a trend. With the exception of the refactoring spike in the attributes and association metamodel, the trend would have been also evident in that case too.

Techniques for mining software repositories can recover more trends about the nature and pattern of software development of Umple. The Umple repository and all its versioning history are publicly available for download. This should encourage other independent work to perform such in depth analysis of the repository and report on it.

4.6 Analysis of the use of Umple to develop Umple itself

We now revisit the requirements specified in Section 2, and reflect, in their context, on our experiences developing Umple as an open source project written in Umple itself.

- **Little to no change to existing infrastructure**: At a bare minimum, Umple developers need to simply download a single Jar file (Umple.jar) to use the Umple compiler. This is readily available and works well with other open source infrastructure. Most developers choose to go one step further and work with the Umple Eclipse plugin. However, installing such plugin is a common task to Eclipse-developers.
- **Easy editing paradigm** – textual models: Umple users can use whatever text editor they are familiar with to edit Umple sources. There is no need to use a separate graphical tool to view the Umple-sources in UML notation. Such visualizations can be generated using Umple Online if desired.
- **Versioning, comparing and merging features that operate the same way on model and programming language text**: All Umple sources, whether they include Java or model elements, are represented textually in Umple files and are committed to SVN just like in any other open-source system.
- **Tight integration with code:** In Umple, both model elements and code elements are co-located in the same textual artifact and treated uniformly
- **Semantic completeness**: The Umple compiler completely supports the subset of UML semantics, like multiplicity constraints, bidirectional association referential integrity, and state-machines, expressible in Umple.

5 Related work

Robbins [6] identifies a set of tools that are absent in an open source community, including requirements and modeling tools. He attributes this absence to a number of reasons, including the fact that open source developers do not have direct paying customers and therefore do not feel the need to follow a rigorous methodology or respect deadlines. He also predicts that many of those missing tools will have a significant positive impact if and when they get adopted by the open source community. This is where the motivation for the research presented in this paper comes from.

Iivari [18] explored why modeling tools are not used in general in software projects. In a study, he reports that 70% of modeling tools are never used after being available for one year, 25% are used by only a single group, and only 5% are widely used. This is despite the fact that the same study reports positive impact of modeling adoption. More than a decade later, modeling tools are still not widely adopted and a significant segment of the software engineering practices are still code centric [3].

6 Conclusion and future work

The open source community remains almost entirely code centric. Adoption of modeling practices remains extremely low. Since the open source community is known for its exploration, innovation, and willingness to quickly adopt new technologies, model-oriented programming languages like Umple offer a promising direction. We have presented Umple as a case study of an open source project that has been successfully developed using a model-based programming language (Umple itself). It effectively supports modeling by embedding model elements in code artifacts, and hence in the same repository. Besides using the Umple compiler, which can be invoked from a regular command line shell and from the Eclipse platform, Umple developers only use openly available tools..

The key concept in our approach is the view that model and code can be treated uniformly. While most models are best viewed visually and most code is best viewed textually, the underlying representation of both model and code should be textual, and capable of co-existing in the same textual artifacts. This helps bring modeling to the open source community. To the best of our knowledge, Umple remains the only open source project that is largely model-driven, and where model contributions are performed routinely.

Future work includes more in-depth analysis of the trends of comments of the Umple open source code. Approaches such as mining software repositories can be utilized to uncover trends and patterns.

Future enhancements to Umple include a tool to support incremental reverse engineering from existing systems (umplification) [19]. The key advantage is that systems can be reverse-engineered while being able to instantiate a running system at all stages. Umple is also being extended to include a debugger that can work at the modeling level. This stems from our vision that whatever functionality available to traditional code developers should also be made available when modeling support is added.

References

[1] Ye, Y. and Kishida, K. "Toward an Understanding of the Motivation of Open Source Software Developers," in *Software Engineering, 2003. Proceedings. 25th International Conference on,* 2003. pp. 419-429.

[2] Selic, B. "Models, Software Models and UML". *UML for Real,* pp. 1-16, 2004.

[3] Forward, A., Badreddin, O. and Lethbridge, T. C. "Perceptions of Software Modeling: A Survey of Software Practitioners," in *5th Workshop from Code Centric to Model Centric: Evaluating the Effectiveness of MDD (C2M:EEMDD),* 2010. Available: http://www.esi.es/modelplex/c2m/papers.php

[4] Ohlo. " The Open Source Network", accessed 2013, http://www.ohloh.net/.

[5] Google Inc. " Chromium", accessed 2013, http://www.chromium.org/Home.

[6] Robbins, J. "Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools". 2005. *Perspectives on Free and Open Source Software, MIT Press,* pp. 245-264.

[7] Hertel, G., Niedner, S. and Herrmann, S. "Motivation of Software Developers in Open Source Projects: An Internet-Based Survey of Contributors to the Linux Kernel". 2003. *Research Policy,* vol 32, pp. 1159-1177.

[8] Badreddin, O., Forward, A. and Lethbridge, T. C. "Model Oriented Programming: An Empirical Study of Comprehension". 2012. *Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research,* pp. 73-86.

[9] Lethbridge T.C., Forward, A. and Badreddin, O. " Umple Language Online.", accessed 2012, http://try.umple.org.

[10] The Eclipse Foundation. " Package Org.Eclipse.Emf.Ecore", accessed 2010, http://download.eclipse.org/modeling/emf/emf/javadoc/2.5.0/org/eclipse/emf/ecore/package-summary.html#details.

[11] Lethbridge, T. C., Forward, A. and Badreddin, O. "Umple Google Code Project". 2012. Available: code.umple.org

[12] O. Badreddin. (2012) "An Empirical Experiment of Comprehension on Textual and Visual Modeling Approaches". University of Ottawa, Available: http://www.site.uottawa.ca/~tcl/gradtheses/obadreldin/

[13] Badreddin, O. "A Manifestation of Model-Code Duality: Facilitating the Representation of State Machines in the Umple Model-Oriented Programming Language". 2012.

[14] Badreddin, O. and Lethbridge, T. C. "A Study of Applying a Research Prototype Tool in Industrial Practice," in *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering,* 2010. pp. 353-356.

[15] Badreddin, O. and Lethbridge, T. C. "Combining Experiments and Grounded Theory to Evaluate a Research Prototype: Lessons from the Umple Model-Oriented Programming Technology".

[16] Verschuren, P. and Hartog, R. "Evaluation in Design-Oriented Research". 2005. *Quality & Quantity,* vol 39, Springer. pp. 733-762.

[17] Atlassian. " FishEye", accessed 2013, http://www.atlassian.com/software/fisheye/overview.

[18] Iivari, J. "Why are CASE Tools Not used? Communications of the ACM". 1996. *Communications of the ACM,* vol 39, pp. 94-103.

[19] Lethbridge, T. C., Forward, A. and Badreddin, O. "Umplification: Refactoring to Incrementally Add Abstraction to a Program," in *Working Conference on Reverse Engineering,* 2010. pp. 220-224.