Combining Experiments and Grounded Theory to Evaluate a Research Prototype: Lessons from the Umple Model-Oriented Programming Technology

Omar Badreddin
School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Canada
obadr024@uottawa.ca

Timothy C. Lethbridge
School of Electrical Engineering and Computer Science
University of Ottawa
Ottawa, Canada
tcl@site.uottawa.ca

***Abstract*—Research prototypes typically lack the level of quality and readiness required for industrial deployment. Hence, conducting realistic experimentation with professional users that reflect real life tasks is challenging. Experimentation with toy examples and tasks suffers from significant threats to external validity. Consequently, results from such experiments fail to gain confidence or mitigate risks, a prerequisite for industrial adoption. This paper presents two empirical studies conducted to evaluate a model-oriented programming language called Umple; a grounded theory study and a controlled experiment of comprehension. Evaluations of model-oriented programming is particularly challenging. First, there is a need to provide for highly sophisticated development environments for realistic evaluation. Second, the scarcity of experienced users poses additional challenges. In this paper we discuss our experiences, lessons learned, and future considerations in the evaluation of a research prototype tool.**

***Keywords*-UML, Modeling, Umple, Empirical Studies, Grounded Theory.**

I. INTRODUCTION

The evolution of programming languages has exhibited a continuous trend of increasing levels of abstraction. Only 60 years ago machine code was used, followed quickly by assembly languages that provided a higher level of abstraction to facilitate the creation and editing of larger code segments. The trend continued to procedural programming languages, and then object orientation. UML adopted an even higher level of abstraction and emerged as the defacto standard for representation of diagrammatic software engineering models. It has been shown that in certain contexts, using such models gives favorable results in improving the efficiency and productivity of software engineers [1, 2].

Despite the demonstration of certain visual modeling benefits, there remains resistance to use of diagrammatic approaches [3], and additional evidence is needed to show they are effective. For example, there are a number of studies that indicated gains in the comprehensibility when using certain visualizations while others have reported that graphics were significantly slower than text in the experimental comprehension tasks [4].

Motivated by deficiencies in the visual modeling paradigms, a number of approaches have incorporated visual abstractions within textual environments. Object Management Group (OMG), the organization that manages the UML standards, has proposed in the past HUNT, a textual notation for UML class diagrams [5]. More recently, Alf [6], a concrete textual syntax for UML action semantics, has been proposed. A language like Ruby supports some of the UML modeling abstractions (e.g. state machines) and makes them available in the language itself [7]. Umple [8, 9, 10], the technology developed by our laboratory at the University of Ottawa, is a model-oriented language where modeling abstractions are embedded in code. Other textual modeling approaches includes MetaUML [11], yUML [12], TextUML [13], State Machine Compiler (SMC) [14], AsmL [15], and Executable UML [16].

The literature is rich with theoretical analytical work on comprehension of notation. The Cognitive Dimensions framework [17], for example, provides valuable perspective on notation and comprehension. This theoretical work is important, but does not reduce the need for realistic experimentation that reflects real life modeling and development tasks.

Our objective is to conduct experiments that reflect real-life tasks, which include manipulation of relatively complex models and systems. Like many researchers, we aim to produce results that can raise confidence and are relevant to professionals and early adopters. In our efforts to evaluate the Umple technology, we embarked on an action research project and hoped to deploy the technology in an industrial project where professionals are accompanied by a researcher. This approach quickly failed; developers had entrenched ways of doing things and were not willing to invest time in trying out a research prototype. Like many research prototypes, Umple was at that time rather 'raw' and had some important weaknesses. Quickly we realized the need to ensure Umple tools gave developers the same experience as any other comprehensive development environment.

University-based research teams lack both the resources to deliver quickly or fully on this kind of objective, although it remains for us a long-term goal. To evaluate our work in the interim, we therefore resorted to other evaluation methodologies that we discuss in the following two sections. We first introduce Umple, the technology under evaluation.

II. BACKGROUND: THE UMPLE TECHNOLOGY

Umple is a model-oriented programming technology. Its key element is a set of language extensions that enhance existing languages like Java, PHP, and Ruby with modeling constructs. It supports UML class diagrams, UML state machines, selected software patterns and code tracing. Umple is currently in use by students and in classrooms. The Umple web-based modeling tool has about 2,000 visits per month [8]. Below is a simple example of Umple that shows a simple state machine model.

```
class Course {
  Boolean maxStudentsReached;
  courseId;
  status {
    Open {
     register-> /{notification();} Closed;
    Closed {
     deRegister -> /{deRegister ();} Open;
     }
    }
  // placeholder methods
  public void notification() { }
  public void deRegister() { } }
```

Fig. 1. Umple example

The Course class has three attributes, *maxStudentReached*, *courseId* and *status*. The status attribute value is controlled by a state machine that has two states, *Open* and *Closed*. When the event *register*() occurs, and if the guard [*maxStudentReached*] is true, then the transition action *notification*() is executed, and the state of the attribute status is updated to *Closed*. When in the *Closed* state, the state machine responds to the event *deRegister*(). Note that we only provided placeholders for the native code that handles notifications and deregistering students.

Umple also supports nested and concurrent states, associations, attributes and model-level tracing. More discussion regarding Umple is available in [18, 19].

III. GROUNDED THEORY STUDY OF UMPLE USERS

The main objective of the study is to guide the development of Umple by users' feedback and utilize the users' perception and experiences to build theories about how textual modeling is perceived and used in practice.

A. Methodology

Our methodology is to conduct questionnaires and interviews with users who have previous experience with Umple. The questionnaire results are analyzed quantitatively. The interview results are analyzed qualitatively following a grounded theory approach.

Participants are asked to fill a questionnaire of 21 questions and are interviewed for about 30 minutes each. The questionnaire and interview questions are related to their experience using Umple, major challenges, and code generation experience. The interview questions are open ended to foster contribution from participants.

B. Results

The results of this study indicate that users perceive textual modeling to be of less complexity than the equivalent Java code. Users also shared their view on the generated code, the textual modeling paradigm, and other aspects of the platform such as error reporting and syntax highlighting. The results of this study were used in two ways; first, to determine a few high level objectives against which design decisions were made. For example, one objective is to reduce the number of keywords in the language on the assumption that fewer keywords results in a language that is less complex and easier to learn. Second, the results were used to incrementally refine the implementation of the language. Users' feedback on the syntax, for example, was taken into consideration.

C. Lessons Learned

One major obstacle we faced in this study is the availability of participants with extensive experience with Umple. Having emerged from the social sciences, the grounded theory approach implicitly assumes the availability of participants who are intimately familiar with the phenomenon under study. This assumption does not hold for experimental development and research prototypes in software engineering particularly when the tool under study has not witnessed popular adoption.

We also identified the need for additional non-traditional data sources. Grounded theory studies typically rely on interviews and/or observation of participants. Interviewing software engineers is more difficult than interviewing a sample of the general public for many reasons; 1) software engineering employees are a volatile group and their participation can be limited by management and confidentiality considerations. 2) There is a small number of software engineers familiar with the tool or phenomenon under study. These suggest that in the process of designing a grounded theory study, data sources other than interviews and observations must be considered. For example, UmpleOnline automatically saves models that have been created or edited online. Such models could be used as a source of data for future studies provided the permission of users was obtained at the time they saved their data.

Another challenge in using grounded theory is background knowledge. It is widely accepted that a researcher should minimize reading of background literature to avoid risk of outside influence and to ensure that theories emerge from within the data. This aspect is difficult to ensure in the software engineering domain. Background knowledge is required for the proper execution of the study.

IV. CONTROLLED EXPERIMENTATION

Following completion of the grounded theory study, we enhanced Umple based on the findings of that study. At this point we were ready to conduct formal experiments.

In our first experiment our goal is to evaluate the comprehension of the Umple textual modeling technology

in comparison with UML and Java. This experiment seeks to validate the hypothesis that Umple has retained the advantages of UML with respect to comprehension. The experiment does not assess whether Umple has also retained any advantages of textual programming. The experiment scope is limited to comprehension and ignores other aspects of development tasks, such as model manipulation, editing, layout, and refactoring. The experiment design specification and results are published in a technical report [20]. Following, we give a brief background on the experiment design, instrumentation, and results.

A. Experiment Design

The controlled experiment treatment is the modeling notation with three possible values: UML, Java, and Umple. Each participant was presented with an instance of each modeling notation. To avoid the learning effect, variations were applied to each modeling instance. The set of experimental objects consisted of nine artifacts, which were comprised of three example systems of comparable complexity, written in three notations (UML, Java, and Umple).

The experiment focused on the following main hypothesis:

H1: A system written in Umple is more comprehensible than an equivalent Java implementation of the system. In other words, participants take on average less time to respond to questions when presented with an Umple version of a system as opposed to a Java version.
The corresponding null hypothesis is:
H1o: Umple and Java do not differ in comprehensibility.

B. Instrumentation

The main experiment instruments were three rounds of comprehension questions that measure the effectiveness of the notation. Each round contained 12 questions. Participants usually provided responses within 30 seconds of posing the question. Some questions addressed the concept of associations as present in a class diagram or a textual notation. Other questions addressed comprehension of a state machine.

The questioning sessions were audio recorded. Time was measured starting from the end of posing a question until the participant correctly answered the question. At the onset of the experiment, participants were asked a number of profiling questions about their background, prior knowledge of UML, Java and Umple, software engineering courses and work experience.

C. Results

Each participant provided answers to 36 questions; 12 answers for each notation. The overall average time to answer the questions for UML was 3.6 seconds, Java was 6.9, and Umple was 3.6 seconds. Using a two-tailed t-test to measure the statistical significance, the comprehension time required for Umple is lower than that of Java ($p=1.5x10-8$). So we reject null hypothesis H1o. Using the Mann-Whitney test (U-test), Umple is still better than Java ($p = 8.9x10-9$) and a W value of 2722. So using this test we also reject the null hypothesis H1o. Using the sign-test, Umple was better than Java in 83 occurrences, while Java was better than Umple in 13 occurrences. The sign test results indicate Umple is better than Java ($p=6.0x10-14$), again leading to rejection of null hypothesis H1o.

Using a two-tailed t-test to measure the statistical significance, Umple does not have a significantly different average comprehension time than UML ($p=0.9$). Using Mann-Whitney test (U-test) Umple is not significantly better than UML ($p = 0.2$) and a W value of 4477.5. Using the sign-test, Umple was better than UML in 53 occurrences, while UML was better than Umple in 30 occurrences. The sign test results also indicate Umple is not significantly better than UML ($P=0.864$).

We also conducted mean and standard deviation analysis. For each participant's results, we test to see if the mean comprehension time of Umple lies in the range of the mean of UML, plus or minus one standard deviation. The answer was positive in all nine participants' results. This technique is used to show whether or not two data sets come from different populations [21]. Here, we use it to show that the two data sets (Umple and UML) are not significantly different, so we cannot conclude that they come from different populations.

D. Threats to Validity

There is a threat that expertise and background of the participants may have an impact on how fast they respond to questions. To mitigate this risk we collected profiling information to validate our claim that our subjects have backgrounds that were not significantly biased towards a specific modeling notation. We also used straightforward modeling examples and questions. This has the effect of shifting the focus on the notation, rather than the subject's technical expertise.

There is an external validity threat that the example systems used in the experiment are not a good representation of the real software engineering systems.

There is a threat to internal validity that the specific questions or example systems may have an impact on the time participants take to respond. This was mitigated by the distribution of participants and treatments, piloting the study and obtaining independent reviews of prior to the study.

E. Lessons Learned

The choice of a controlled experimentation proved to be effective in handling the following aspects:

- Immaturity of the Umple development platform: Umple has been developed following a strict test-driven approach [22]. However, Umple is not a commercial product and lacks some of the features available in the development environments of other high level programming languages, such as code-assist, sophisticated refactoring, etc. Umple also has only

limited support for error handling and error messages. Using a controlled experiment, we were able to evade such limitations and focus on core modeling paradigms.

- The ability to measure comprehensibility in isolation of other tasks: The designed experiment enabled our team to evaluate comprehension of the notation in isolation of other tasks, such as model editing or system development.

V. Future Work

We believe that if and when Umple witnesses popular uptake and resources become available, a more extensive grounded-theory study would yield a wide range of useful data to improve Umple. Such a study would focus on the experiences of people using Umple naturally as part of their everyday work in industry.

Controlled experimentation based on a narrow set of hypotheses has given us evidence that Umple is a useful tool. We plan to conduct a additional controlled experiments to test a variety of other hypotheses. Our plan is as follows:

Since an experiment does not require any background knowledge of Umple, we plan to recruit professional participants who would not have had to spend time developing in Umple (unlike in a grounded-theory study). First, we can confirm or reject earlier results; second, we can contribute to answering the long standing question of whether student subjects results are comparable to professional subjects or not [23]. Third we can enhance the experiment's example systems with larger and more realistic models. We also plan to arrange for developers to create small systems using either Umple or UML diagrams, and measure productivity and the quality of the resulting systems.

VI. Conclusion

This paper reported on our efforts to empirically evaluate a research prototype tool. We believe that many researchers in the software engineering domain face similar challenges where they want to evaluate *not-so-fully-functional* experimental tools. The software engineering research domain is behind the social sciences in the use of empirical qualitative research methodologies such as grounded theory. Such research methodologies may not be well adapted to many evaluation tasks in the software engineering domain. Controlled experimentation is a good alternative, but one that usually suffers from external validity threats. We believe that combining the two types of studies is a good overall approach.

References

[1] Grossman, M., Aronson, J. E. and McCarthy, R. V. "Does UML make the Grade? Insights from the Software Development Community". 2005. Inf Software Technol, vol 47, pp. 383-397.

[2] Anda, B., Hansen, K., Gullesen, I. and Thorsen, H. "Experiences from Introducing UML-Based Development in a Large Safety-Critical Project". 2006. Empirical Software Engineering, vol 11, pp. 555-581.

[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] Hendrix, D., Cross II, J. H. and Maghsoodloo, S. "The Effectiveness of Control Structure Diagrams in Source Code Comprehension Activities". 2002. IEEE Trans.Software Eng., Published by the IEEE Computer Society. pp. 463-477.

[5] Object Management Group (OMG). "Human-Usable Textual Notation", accessed 2012, http://www.omg.org/technology/documents/formal/hutn.htm.

[6] Mentor Graphics Corporation. "Concrete Syntax for a UML Action Language, Action Language for Foundational UML", accessed 2012, http://lib.modeldriven.org/MDLibrary/trunk/Applications/Alf-Reference-Implementation/doc/.

[7] Ruby Language. "Rails State Machine". Version 5.0. Accessed 2012. Available: http://www.ruby-lang.org/en/

[8] Complexity Reduction in Software Engineering Group of the University of Ottawa, " Umple Online.", accessed 2012, http://try.umple.org.

[9] Lethbridge, T., Mussbacher, G, Forward, A. and Badreddin, O, (2011) "Teaching UML Using Umple: Applying Model-Oriented Programming in the Classroom", CSEE&T 2011, pp. 421-428.

[10] Forward, A., Badreddin, O., Lethbridge, T.C., Solano, J., (2011) "Model-Driven Rapid Prototyping with Umple", Software Practice and Experience, DOI: 10.1002/spe.1155

[11] MetaUML, U. Diagrams for LaTEX, Internet,

[12] Harris, T. " YUML", accessed 2012, http://yuml.me/.

[13] Chaves, R. " TextUML", accessed 2012, http://abstratt.com/.

[14] Charles W. Rapp. "The State Machine Compiler". vol. 6.0.1, December, 2009.

[15] Gurevich, Y., Rossman, B. and Schulte, W. "Semantic Essence of AsmL". 2005. Theor.Comput.Sci., vol 343, Elsevier. pp. 370-412.

[16] Raistrick, C., Francis, P. and Wright, J. Model Driven Architecture with Executable UML (TM). Cambridge University Press New York, NY, USA, 2004.

[17] Green, T. R. G. (1989) Cognitive dimensions of notations. In A. Sutcliffe and L. Macaulay (Eds.) People and Computers V. pp 443-460. Cambridge: Cambridge University Press.

[18] Lethbridge, T.C., Badreddin, O. "Umple - Associations and Generalizations". Instructional video. vol. youtube video, 2011. Available http://www.youtube.com/watch?v=HIBo0ErCVtU

[19] Badreddin, O. "Umple: A Model-Oriented Programming Language," in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, 2010. pp. 337-338.

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

[21] Mohammad, S. "From Once upon a Time to Happily Ever After: Tracking Emotions in Novels and Fairy Tales". 2011. ACL HLT 2011, pp. 105-114.

[22] Gupta, A. and Jalote, P. "An Experimental Evaluation of the Effectiveness and Efficiency of the Test Driven Development". Proc. First International Symposium on ESEM, IEEE, 2007, pp. 285-294.

[23] Höst, M., Regnell, B. and Wohlin, C. "Using Students as subjects—a Comparative Study of Students and Professionals in Lead-Time Impact Assessment". 2000. Empirical Software Engineering, vol 5, Springer. pp. 201-214.