CAPSTONE SOFTWARE ENGINEERING STUDENTS CAN DEVELOP A HIGH-QUALITY COMPLEX SYSTEM: A CASE STUDY WITH UMPLE

Timothy C. Lethbridge
University of Ottawa
Timothy.Lethbridge@uottawa.ca

Abstract – Umple is an open-source programming technology developed almost entirely by students, the majority of whom were working on it as their capstone project through a program called UCOSP. We describe our development process for Umple that has provided a rich educational experience for the students, while at the same time continually improving Umple’s quality. We also describe features of Umple that have been designed to facilitate its use in teaching software engineering.

Keywords: Capstone project, open source, software engineering, collaboration, model-driven development

1. INTRODUCTION

We describe our approach that has enabled successive groups of students to engineer and evolve a complex, high-quality software system, while serving as an educational platform for them and for other end-users.

Umple is a tool for engineering software using a combination of high-level abstractions (class diagrams, state diagrams), traditional code, and several other innovative mechanisms. We describe Umple in more detail in Section 2.

Umple has been developed by students, for students. It is both a research project and an educational project. Many research tools fail to achieve the levels of usability, reliability and maintainability needed for widespread adoption and long-term use. A key objective in Umple, has been to demonstrate that this need not be the case, while providing an excellent educational opportunity both for student developers of Umple and for worldwide student users of Umple.

2. OVERVIEW OF UMPLE

Umple [1][2] is a compiler supporting high level abstractions including state machines, associations, traits, constraints and aspects. Method bodies and other algorithmic code can be written in Java, C++ or PhP. Code generated from the high-level abstractions can be generated in these same programming languages.

Umple can be run Online, in Eclipse, in Visual Studio Code and in several other tools. It can also be used in a classic manner as a command-line tool.

Umple is targeted for educators who teach modeling [3][4], for students, and for open source developers who want to build systems that require low coding effort. The core idea is that the high-level abstractions reduce the amount of boilerplate (repetitive) code that needs to be written, while at the same time enabling developers and students to think at a higher level and to visualize their code. Umple is a textual language, but its high-level abstractions can also be rendered and edited as graphs.

Umple’s online user manual [5], if printed, amounts to 300 pages. The OpenHub code analysis system [6] points out that 102 people have contributed a million lines of code to Umple and suggests they have spent 286 person-years of effort on it based on its apparent complexity. About 10% of the lines of code (about 100K LOC) are the core Umple compiler, written in Umple itself. The remainder includes the UmpleOnline website and an extensive test suite. One thing to bear in mind is that since Umple contains many high-level abstractions, it is designed to be able to represent systems with many fewer lines of code than would otherwise be necessary, so a system with the same capabilities written in a traditional language would be 5-7 times larger in terms of lines of code.

There is not space here to do justice to Umple’s comprehensive feature set. However, the following is a taste.

The example below has three classes starting at lines 1, 5 and 31. Line 6 shows an association, in which a class diagram shown in Figure 1 has been synchronized with the remainder includes the UmpleOnline website and an extensive test suite. One thing to bear in mind is that since Umple contains many high-level abstractions, it is designed to be able to represent systems with many fewer lines of code than would otherwise be necessary, so a system with the same capabilities written in a traditional language would be 5-7 times larger in terms of lines of code.

There is not space here to do justice to Umple’s comprehensive feature set. However, the following is a taste.

The example below has three classes starting at lines 1, 5 and 31. Line 6 shows an association, in which a conference can have any number of papers, but a paper can be in at most one conference. This association generates code that manages bidirectional navigation and referential integrity; in other words the links from ConferencePaper instances to Conference instances are kept properly synchronized with the links in the opposite direction.

Line 8 is another association that has unidirectional navigation. The class diagram shown in Figure 1 is automatically generated from this text in UmpleOnline [16].
A state machine called status is defined between lines 9 and 29. Lines 11, 14, 18, 17 and 20 define events (which generate methods that can be called). Lines 13 to 22 define a state that has sub-states. The state diagram shown in Figure 2 is also displayed when the code below is loaded into UmpleOnline.

```
1 class Conference {
2     name;
3     Date startDate;
4 }
5 class ConferencePaper {
6     * -- 0..1 Conference;
7     title;
8     0..1 -> 1.* Person author;
9     status {
10        inPreparation {
11            submit -> underReview;
12        }
13        inConferencePipeline {
14            withdraw -> withdrawn;
15            underReview {
16                accept -> accepted;
17                reject -> rejected;
18            }
19            accepted {
20                publish -> published;
21            }
22        }
23        rejected {
24        }
25        withdrawn {
26        }
27        published {
28        }
29    }
30 }
31 class Person {
32     name;
33 }
```

Umple has been funded by several sources: One of its original sponsors was IBM, who were interested in the potential of a textual syntax for UML. Additional funding has come from collaborations with General Motors, who were interested in Umple’s real-time capabilities. Government funding has come from NSERC (Discovery Grants) and the Ontario Research Fund. Development has been ongoing since 2007 and is expected to continue.

Some of Umple’s advanced features include its support for three kinds of separation of concerns mechanisms, aspects, traits [9] (an ability to specify modeling elements and code that can be injected into multiple classes regardless of the inheritance hierarchy), and mixsets [10] (an ability to specify features or variants of product lines that can be annotatively blended with code, or kept in separate files). Umple also has its own parsing technology and built-in textual template capability [11]. All these features add to the richness of what can be taught using Umple, as well as what can be developed with Umple.

Usability of Umple has been a priority. This has been tested in several experiments [17]. Maintainability of Umple has also been central, and has been one of the enabling qualities for Umple to be easily modified by capstone students: Its use of test-driven development, model-driven development and innovative separation of concerns mechanisms has facilitated this maintainability. Documentation of the Umple compiler is auto-generated, which also facilitates maintainability. For example, the documentation of Umple’s internal parser is here [18] and a generated class diagram of the entire system is here [19]. Another contributor to the maintainability to Umple is a rigorous policy of avoiding dependencies (such as a dependency on Eclipse) that are commonly found in other modeling tools.
3. THE NEED FOR UMPLE AS A TEACHING TOOL

The need for Umple was noted by the author after many years of teaching software modeling in an introductory second year software engineering course.

From the decade starting in the mid-1990s, many modeling tools were developed and used in the classroom. But these tools tended to be complex or didn’t allow generation of actual systems. Most tools had both types of weaknesses. For example, ArgoUML [12] and Rational Rose were very popular, but only generated ‘stubs’ of code.

We noted that students would make many mistakes: They would add classes that were not needed; they would mistake operations for classes, such as a ‘Borrow’ class instead of a ‘Loan’ class in a library system. They would create invalid associations, particularly specifying multiplicity wrongly. For the most part they were just using UML as a diagramming notation. We wrote a textbook [15] to try to teach students not to make these mistakes, but without being able to formally analyse the model or actually generate real systems from it, students were not able to get the sort of feedback needed to increase their modeling skill.

Based on the experience described above, we created Umple with several objectives.

• To be able to create real systems with a modeling tool. We wanted to demonstrate that adding modeling constructs from UML and using them for real system development need be no more difficult than using programming language constructs.

• To demonstrate that both text and diagrams are equally useful in software modeling and that the two together operate synergistically.

• To make modeling as simple as possible and in a way that would work parsimoniously with programming languages, a wide variety of integrated development environments (IDEs) and testing tools.

We later on evaluated Umple and it was shown to improve student understanding and grades [2]. In another evaluation, Umple was shown to have many valuable properties as compared to other modeling tools used in education.

4. UMPLE DEVELOPMENT PROCESS IN BRIEF

Umple has been under development since 2007. For the first few years the contributors were mostly graduate students; undergraduate involvement started in 2010. All students (undergraduates and graduates) follow the same process.

Umple has been built by 6 PhD students, several master’s students and over 60 undergraduates doing their capstone projects. Most capstone students were part of the UCOSP program [8] through which students from various Canadian universities participated in open source teams.

Umple has been open source since 2011 [7]. It is written in itself and follows agile software engineering practices, particularly rigorous test-driven development at multiple levels (tests of parsing, of generating valid code and of executing compiled systems), as well as frequent (continuous) releases that require 100% of tests to pass as well as code inspection.

Umple follows model-driven development practices, meaning that the model, in the form of class diagrams and state diagrams, is the core of the system. This is made possible by Umple being written in itself.

UCOSP involvement started in 2011. This is described in the next section.

5. UCOSP AND UMPLE

UCOSP (Undergraduate Capstone Open Source Projects) [8] [9] operated from 2008 to 2018 working with groups of students every autumn and winter. Umple was involved from 2011 onwards. The general process for each semester’s UCOSP session has been as follows:

The UCOSP steering committee solicited mentors running open source projects and also contacted professors of software engineering and computer science capstone courses in Canadian universities. The professors were asked to propose one or two of their best capstone students.

The author of this paper has participated both as an open-source project mentor, since he runs the Umple project and as capstone course professor. In the latter role, he has arranged for several University of Ottawa students to participate in non-Umple projects (e.g. ReviewBoard, Firefox, or Blackberry open source tools). The focus of the current paper is the former role, where he has helped mentor students from other universities who are working on Umple.

Students worked on their UCOSP project as a single-semester course, so their workload is about 10-12 hours a week for about 12 weeks.

Over the years, each Umple UCOSP cohort has had between 4 and 5 students, and they came from a wide variety of Canadian Universities. The process of working with an Umple UCOSP group was as follows:

Students were first introduced to Umple’s philosophies, architecture and development process through various tutorials.

Students were then tasked to set up their development environment and build Umple [13], as well as set up a local UmpleOnline web server for their own testing purposes.

They were then tasked with fixing a small bug in the system. After about 15 days of work (e.g. around January 20 or September 20) students would attend an in person 3-
day code sprint. Over the years these have been held in cities such Vancouver, Toronto and Ottawa. Student travel has been covered by a grant from Google (and once from Facebook, when it was held in California). The mentor and students worked together for about 20 hours, by the end of which most students were able to successfully contribute at least one pull request to fix their initially-assigned bug or make some other improvement.

After fixing a few bugs, students usually went on to make more substantive contributions during the semester.

UCOSP students have solved hundreds of issues in Umple, and made some major contributions, including generation of Entity-Relationship diagrams, converting Umple internally to use its own UmpleTL templating [11], making UmpleOnline work in Docker, making associations immutable and many more.

The UCOSP program has been suspended as of early 2019, but we hope it will resume, or that a similar program be established. The reasons for its suspension stem from the labour involved in mentoring. The lead author was able to sustain the mentoring work on Umple, along with his PhD students. However, there was a need for a wide variety of other open source projects to also participate, and this sometimes proved challenging as most mentors of open source projects were not actually paid to work on those projects.

6. LEARNING OUTCOMES OF CAPSTONE STUDENTS WORKING ON UMPLE

Benefits students have received in their capstone work on Umple include exposure to development of a real compiler, a large codebase that honed code exploration and debugging skills, a multi-layer testing architecture and practical UML modeling.

Contributions to the compiler require that all 6000+ tests pass, that new tests be written covering the contribution, and documentation be updated (user manual, code comments and architecture wikis). Capstone students thus benefit from exposure to practices that train them well for industry, and also from interacting with graduate students working on model-driven development.

The following are some of the specific learning outcomes from the participation of capstone students in Umple

• Ability to follow standard agile methods including test-driven development and continuous integration in an open source context.
• Ability to understand and correctly modify code in a code base of over a million lines of code.
• Ability to perform model-driven development
• Ability to work with grammars and parsers
• Ability to work with code generators

7. LEARNING OUTCOMES FROM STUDENTS USING UMPLE IN OTHER COURSES

Umple has been designed to facilitate teaching UML and Modeling, and has been used for this from high-school to graduate school in many types of courses.

UmpleOnline processes over 50,000 transactions in an average month (and sometimes on peak days over 10,000 transactions). It hosts an average of 200,000 visitor sessions a year. These statistics testify to its wide use among students, but don’t include those who have downloaded the compiler, or are using its Docker image.

The following are some of the learning outcomes that can be achieved by students using Umple:

• Ability to analyse, debug and generate code from models consisting of class diagrams and state machines
• Ability to create complete executable systems following a model-driven engineering process.
• Ability to use modeling in an agile context.
• Ability to use traits, aspects, mixins and other separation-of-concerns mechanisms.
• Ability to write tests that operate on models.
• Ability to use version control on models

8. CONCLUSIONS

In this paper we have provided an overview of how Umple has been used in the education of software engineering students at all levels. Umple can be used to teach many software engineering concepts both in ordinary courses requiring teaching of modeling, and also in Capstone courses, where students can contribute to the Umple compiler or its website.

Acknowledgements

We thank IBM, Google, Facebook, NSERC, all of whom have provided direct or indirect funding. We also thank the UCOSP steering committee and all Umple contributors.

References

[1] Umple home page, http://www.umple.org. Accessed January 2019
[2] Lethbridge, T.C., Abdelzad, V., Husseini Orabi, M., Husseini Orabi, A., Adesina, O., (2016) “Merging Modeling and Programming using Umple”, ISOLA 2016, Corfu, Springer LNCS 9953, pp. 187–197
[3] Lethbridge, T.C. (2014), “Teaching Modeling Using Umple: Principles for the Development of an Effective Tool”, CSEE&T 2014, IEEE-CS, pp 23-28.
[4] Agner, L.T.W., Lethbridge, T.C., Soares, I.W. (2019) “Student Experience with Software Modeling Tools”, SoSyM, Springer, https://rdcu.be/bfxpo

[5] U ample User Manual, http://manual.umple.org, Accessed January 2019

[6] Open Hub Umple, https://www.openhub.net/p/U ample, Accessed January 2019

[7] Github Umple, https://github.com/umple/umdle, Accessed January 2019

[8] Holmes, R., Craig, M., Reid, K., and Stroulia, E., “Lessons learned managing distributed software engineering courses”, ICSE Companion’14

[9] Abdelzad, V, and Lethbridge, T.C. (2015) “Promoting Traits into Model-Driven Development”, Software and Systems Modeling, 16:997–1017

[10] Lethbridge, T.C. and Algablan, A. (2018) “Using U ample to Synergistically Process Features, Variants, U ML Models and Classic Code”, ISOLA, October, Springer, LNCS 11244, DOI: 10.1007/978-3-030-03418-4_5 pp. 69-88

[11] H usseini-Orabi, M., H usseini-Orabi, A., and Lethbridge, T.C. (2019) “Umple as a Template Language (Umple-TL)”, Modelsward 2019, pp. 98-106 DOI: 10.5220/0007382000980106

[12] Ramirez, A., Vanpeperstraete, P., Rueckert, A., Odutola, K., Bennett, J., Tolke, L. and van der Wulp, M., 2011. ArgoUML User Manual. Technical report. http://openstorage.gunadarma.ac.id/mirrors/gentoo/distrofiles/argomanual-0.26.pdf

[13] U ample Development Setup, https://github.com/umple/umdle/wiki/DevelopmentSet Up

[14] UCOSP website: http://ucosp.ca

[15] Lethbridge, T.C. and Laganière, R. (2004). Object Oriented Software Engineering: Practical Software Development Using U ML and Java, Second Edition, McGraw Hill: Maidenhead, UK.

[16] UmpleOnline: http://try.umple.org

[17] Badreddin, O. and Lethbridge, T. (2012) “Combining Experiments and Grounded Theory to Evaluate a Research Prototype: Lessons from the Umple Model-Oriented Programming Technology”, USER 2012, in conjunction with ICSE, pp 1-4. DOI: 10.1109/USER.2012.6226575.

[18] U ampleInternalParser documentation http://cruise.eecs.uottawa.ca/umple/umple-compiler-javadoc/index.html?cruise/umple/compiler/UmpleIn ternalParser.html

[19] Umple metamodel : http://metamodel.umple.org