# Teaching UML Using the Umple Model-Oriented Programming Technology

Timothy C. Lethbridge, Gunter Mussbacher, Andrew Forward, Omar Badreddin
*University of Ottawa, Canada*
*tcl@site.uottawa.ca, gunterm@site.uottawa.ca, aforward@gmail.com, obadr024@uottawa.ca,*

***Abstract***

*We show how a technology called Umple can be used to improve teaching UML and modeling. Umple allows UML to be viewed both textually and graphically, with updates to one view reflected in the other. It also allows UML concepts to be added to several base programming languages and web-based code generation from UML to those languages. We have used Umple in student laboratories and homework assignments for two years, and also 'live' in the classroom. In response to a survey, students showed enthusiasm about Umple, and indicated they believe it helps them understand UML better. Improvements in their grades also provide evidence supporting our approach.*

## 1. Introduction

Modeling, particularly using UML, is a promising avenue leading towards greater quality and productivity in software engineering. Unfortunately, it is difficult to educate students to create good UML models, and most industrial practitioners use models only informally [1, 2].

In this paper we will explain how a technology called Umple helps students grasp UML more quickly. In Section 2, we give an overview of Umple. Sections 3 and 4 discuss how we have used Umple in the classroom, in student laboratory sessions and student assignments. Section 5 presents a survey showing that students are very enthusiastic about Umple, and Section 6 presents an analysis of grades, showing that students' ability to create UML class diagrams has improved considerably since Umple was introduced.

### 1.1 Early experience teaching modeling

The first author has taught class diagram modeling in an introductory software engineering course for almost 20 years, first using Rumbaugh's OMT technique [3] and since the mid 1990's using UML. In the early years, upon marking student work and exams, the first author noticed patterns of misconception. The most common were i. inappropriate or missing generalizations, ii. conflating instances with classes, iii. poor use of or omission of multiplicity, iv. missing associations, v. use of associations to represent actions, vi. including both an association and an attribute for the same thing, and vii. including elements in both a class and its sub-classes. A large percentage of students would answer a class-diagram question by drawing lines between classes without showing that they understood the semantics or pragmatics of what they were drawing. Many students would also make major syntax errors, such as putting multiplicity on generalizations.

In response to these and other problems, the first author crafted teaching materials and exercises to specifically train students in good modeling practices, and practices to avoid. Some of the good practices were codified as patterns that went beyond what is found in the

Gang of Four book [4]. In the end, the first author put these all together into a book [5], whose first edition appeared in 2001. Course notes [6] about this material are also available.

## 1.2 Experiences since 2001 leading to the development of Umple

From 2001 on, students did considerably better on the smaller problems, but still seemed to have difficulty making coherent models that could actually be used to build systems involving eight or more classes. The following is an example of a larger modeling problem that students would be trained to solve. A possible answer to the problem appears in Figure 1.

> *Create a class diagram for the following system. Show classes, associations and generalizations: A take-out pizza restaurant wants to set up an online ordering system. A customer must have an account to use the system. When the customer creates his or her account, the following information is stored: Email address (which becomes the user id), contact phone number, password, name, address, preferred credit card number, and credit card expiry date. When the customer creates an order the following information is stored: The time the order was placed, the address for delivery, the contact phone number, the total price, the credit card number charged, the expiry date of the credit card, the items ordered and the total price. An item can be pizza or drinks. For each pizza item, the information stored will include the kind of pizza (thin crust, thick crust or gluten-free crust), the size (small, medium, large), the list of toppings (e.g. cheeze, bacon, vegetables, etc.), the number of items like this (e.g. 10 would mean 10 identical pizzas) and the total price for this pizza item. For each drink item, the information stored is type, size, number, and total price. The system also records each delivery: Associated with each delivery is the name of the delivery driver; the time the driver picked up the order(s) and the time each order was delivered. A driver may take more than one order on a delivery run.*

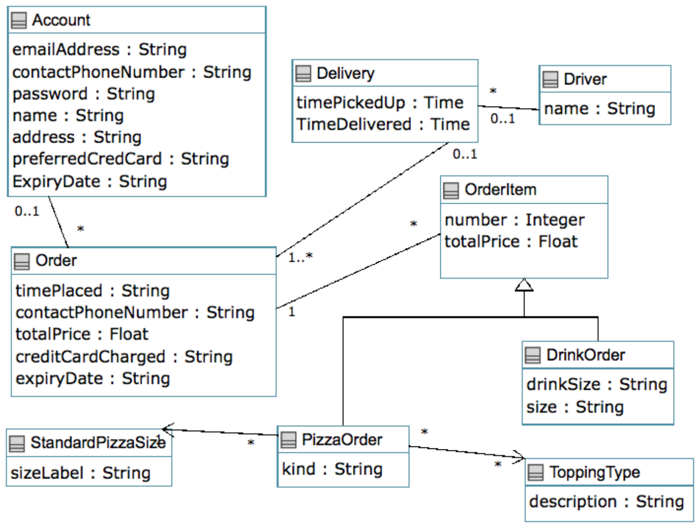

**Figure 1: Answer to typical modeling problem given to students as a homework assignment or exam question**

Students were given laboratory exercises using tools, such as Rational Software Architect, that generate code. Part of the objective was to have them gain experience with tools used in industry. Another part of the objective was to have students learn the consequences of their models, in the same manner that a programmer learns to avoid errors by repeatedly compiling the program, observing it run, and then editing the program to make improvements. Unfortunately, we found that the available tools were unwieldy for students, and often did not generate proper code anyway, especially for state machines. The third author's PhD thesis [7] provides a detailed assessment of the quality and completeness of code generated by a variety of tools.

The first author also wanted a way to rapidly create and edit models in the classroom. His teaching methodology involves extensive board-work exploring modeling alternatives. However for large models, this approach is slow. A lot of erasing and redrawing is necessary as various alternative designs are explored.

He desired a web app that would have no installation footprint, and where the modeling tools would instantly appear. He also wanted to show students, who had already been trained in Java in first year, how easy and quick it could be to use a model to create a running Java program, or to take a Java program and show the model of it. It tended not to be easy to do these two things on the spur of the moment in the classroom due to the awkwardness of tools.

In 2007, the first and third authors conducted an extensive survey of professionals [1, 2]. They also complained about the usability of tools and the difficulties of generating systems from them. We therefore developed the Umple language [8, 9] and set about testing it in the classroom. Umple was has been developed largely by the first, third and fourth authors, plus Dusan Brestovansky [10]. Several other students have used Umple in their thesis work.

## 2. Overview of Umple

Umple adds syntactic elements to certain base programming languages in order to allow UML modeling directly in those languages. The base languages currently supported are Java, Ruby, and PHP, although in this paper we limit the discussion to its use with Java.

We will present an example: If one is creating class Student and class Activity, Umple allows you to specify the following directly in the Java code:

**Associations**: In class Student one can add the declaration `* -- 1..* Activity;` in order to specify a UML association between the two classes. There is now no need to write methods to add links, delete links, or query links of this association; Umple will generate code with an API for doing all necessary operations. Umple supports all UML multiplicity constraints and manages referential integrity; this means, for example, ensuring that if a Student is involved in an Activity, then the Activity object also knows about the Student.

**Attributes**: Attributes in UML and Umple are much more than simple member variables, even though their declarations look very much the same. When one declares an attribute in Umple, one's system then contains a private member variable, generally with methods to get and set it. However depending on which additional keywords one specifies, the attribute could be immutable, it could be required to be unique in each object, or it could be forced to adhere to some various other constraints.

**State machines**: In class Student, one could have a *status* variable with three possible states – initially *entered*, then *registered* as the student registers, and finally *paid* when the student pays. This behavior can be modeled in UML and Umple as a state machine. The Umple code is as follows:

```
class Student {
  status {
     entered { register -> registered; }
     registered { pay -> paid; }
     paid {}
   }
}
```

An Umple program could range from ‘pure model’ as in the above code, to ‘traditional code with a little bit of UML mixed in’, allowing the programmer/modeler to choose from a wide variety of developmental styles. In the teaching discussed in this paper, we have focused on using Umple mostly for pure modeling, although students are asked to write additional code to manipulate the model elements.

Umple runs as an Eclipse plugin and also as a web application called UmpleOnline [11]. The web application allows the user to draw UML diagrams. Umple code appears next to the diagram and is edited automatically as the diagram changes. Alternatively, the user can edit the code and see the diagram change automatically. We encourage the reader to try UmpleOnline before reading on, so as to better appreciate the remainder of this paper. We suggest clicking on ‘Load & Save’ and then selecting one of the Umple examples to explore.

## 3. Use of Umple live in the classroom

The first author has for many years taught two courses that involve modeling. The first (SEG2105) [12] is an introductory second-year course on software engineering that maps to SE201 in the SE2004 curriculum [13, 14]. The first author’s textbook [5] is designed or this course. Students in all computing disciplines, including computer science, software engineering, and computer engineering take it. Students in this course have in recent years had two prior courses in Java, but only a trivial amount of education in modeling using UML.

The second course (SEG4110) [15] is a fourth-year course in advanced design. This course is an elective course for software engineering students, that looks at more esoteric features of UML and much larger design problems. The second author served as a teaching assistant for this course for three years, and developed the laboratory materials, which will be discussed in Section 4.

Prior to the availability of Umple, the first author used a mixture of PowerPoint slides [6, 15] and chalkboard work to teach students how to create the various types of UML diagrams, and also to show the kinds of code that can be written to implement the models. The PowerPoint slides show examples of good solutions to modeling problems, as well as some things to avoid. The board work for class diagrams is usually initiated by presenting a modeling problem as a paragraph of requirements (as was discussed in Section 1.2), and then asking students questions, such as: What classes are central to this problem? What other classes should we add? What attributes should we add and where? And what associations should we add? A similar approach is followed for state diagrams. The fourth-year course also integrates most other UML diagram types.

Starting in 2009, UmpleOnline [11] was introduced into the mix of teaching tools. It did not replace PowerPoint, because there is a benefit of having slides that incorporate a combination of a diagram with explanatory text. In addition, it did not replace board work, because ‘walking around’ the classroom to draw on a large board still helps stimulate students. However, UmpleOnline provided some key benefits to the overall teaching style:

- It proved faster than writing on the board to create diagrams, with its textual mode being subjectively at least twice as fast as writing on the board, and the mouse-and-icon diagram entry mode being about 25% faster than writing on the board[1].

[1] Currently, diagram editing in UmpleOnline is only available for class diagrams. However, code generation is available for both class and state diagrams.

- The ability to call up diagrams from a list of example files and then edit them was something that could not be accomplished readily in PowerPoint nor on the board.
- When designing interactively, the ability to rearrange a diagram was extremely helpful, and much faster than erasing part of a diagram drawn on the board.
- The ability to show several alternative designs quickly for the same problem proved very useful. This ability just requires pasting Umple text or loading pre-defined Umple examples.
- The ability to take an existing Java program and convert it into a model live in front of students showed them that their programming skills can directly connect to their new modeling skills. This is also something that cannot be done with other UML tools.
- The ability to generate and run good-quality code instantly helped students see that modeling is not just about pretty pictures. Doing this helped show them in concrete terms what it means to, for example, change a multiplicity or add an event to a state machine.
- Since UmpleOnline requires nothing but a browser, it can be instantly invoked in any lecture when needed. Furthermore, students can take the same examples and instantly use them on their own laptops, which many now have in the classroom.

UmpleOnline's user interface is simplified, as compared to user interfaces found in most UML tools. This simplification allows the student to focus on the essentials of UML. The first author has tried to use other UML tools live in the classroom, including ArgoUML [16], Rational Software Modeler [17], and the now-obsolete Rational Rose, but these tools did not provide an enjoyable or productive classroom experience. Ultimately, the first author has found board-work to be superior to these tools.

In the Fall 2010 semester, the first author adopted Umple more extensively in the classroom when presenting class and state diagrams. The mix was about 45% of the time using PowerPoint, 30% using board work, and 25% using UmpleOnline. Total time on the topics was considerably reduced from about 15 lecture hours to about 12, because of three factors: a) The speed of displaying and editing new models; b) the ability to show actual generated code rather than explaining abstractly the consequences of various modeling decisions; and c) the ability to respond to student questions in ways that seemed more concrete and straightforward. The time saved was taken about equally from the PowerPoint and board work elements. As we will see in Section 6, the reduced classroom time on the topics did not result in lower grades; in fact, the opposite occurred. The saved time was used later in the course to increase the coverage of design, testing and project management topics.

## 4. Use of Umple for student laboratory exercises and assignments

Students used Umple in laboratory exercises in two courses. In both cases, attendance was compulsory and they had to follow a sequence of written instructions.

### 4.1. Laboratory exercises in the second-year course

Students in the second-year course were given a laboratory exercise in Umple, which can be found online [18]. This exercise involved the following types of steps:

a) Entering a UML model graphically in UmpleOnline, then generating code and looking at the code.

b) Entering another model textually.

c) Making various edits either graphically or textually in UmpleOnline. If students made a mess of their model, they were given textual Umple code they could copy and paste to restore their work to a 'sane' state.

d) Transferring Umple from UmpleOnline to the Umple Eclipse plugin, and writing a main program to test the generated code.

e) Changing the Umple model, and changing the main program to correspond.

Students had to show their results to the teaching assistant at various steps to obtain their grades. They did not have to write a lab report.

### 4.2. Laboratory exercises in the fourth-year course

Students in the fourth-year class were also given laboratory exercises. These exercises included the following types of questions.

a) Explaining the semantics of some snippets of Umple code, or the difference in semantics between two snippets such as the following:

```
class Test {
   defaulted String p = "seg";
}

class Test {
   String p = "seg";
}
```

b) Writing Umple code corresponding to a relatively complex UML diagram they were given, then generating code for this model.

c) Testing the above using JUnit test cases, including some test cases that violate certain UML constraints expressed in Umple.

Students had to turn in their results as a lab report for later marking.

### 4.3. Assignments

In addition to the lab instructions in the second-year course, students also were allowed, but not compelled, to use Umple in homework assignments. They could instead choose to draw diagrams using any other tool, or even by hand. Furthermore, they could generate code from any other tool or manually write it. There were three individual assignments that required drawing UML class and state diagrams, and about half the students chose voluntarily to use Umple for these tasks. Students also worked on a group project, and again, most students generated their code using Umple rather than writing it by hand. A few commented that they had to work around bugs.

In the fourth-year course, a large modeling assignment required students to create an Umple version and generate code in addition to UML models. Some Umple bugs were discovered by the students while working on this assignment.

## 5. Student satisfaction survey

We conducted a survey from a random sample of students in the Fall 2010 offering of the second-year course in which we posed some Likert-scale type questions, plus left space for open-ended comments. 30 students participated, which was a little less than half the class. Students who participated took less than 5 minutes each to answer the questions. Most did it just before or just after one class; no lecture time was consumed in the process. Students were clearly told that completing the survey was voluntary and anonymous.

The following are the results. Averages and standard deviations were calculated by scoring the lowest point on the five-point scales as 1 and the highest point as 5. The first seven questions were to be answered on a scale with values ‘Strongly disagree’, ‘Disagree’, ‘Neutral’, ‘Agree’, and ‘Strongly agree’. Figure 2 summarizes the results.

**Q1. The use of Umple in the classroom by Professor Lethbridge helped me understand UML class diagram concepts.** Average = 4.0 'Agree'; median = 4.0 'Agree'; nobody disagreed.

**Q2. The use of Umple in the lab helped me understand UML class diagram concepts.** Average = 3.8 (A little below Agree); median = 4.0 (Agree); nobody disagreed.

**Q3. I think I will get a better grade in this course because of the use of Umple.** Average = 3.0 (Neutral); median = 3.0 (Neutral). The standard deviation was high (0.91). 37% agreed and 30% disagreed, although none strongly. This result should be contrasted with the results in Section 6.

**Q4. It would be a good idea for Professor Lethbridge to continue to use Umple when he gives this course in future.** Average = 4.0 (Agree); median = 4.0 (Agree); only one person disagreed.

**Q5. As it currently stands, Umple is too incomplete or buggy to be used in this course.** Average = 3.2 (A little above Neutral); Median = 3.0 (Neutral). The standard deviation was high (0.91). 27% disagreed; 10% strongly agreed. Some people commented that the compiler had few bugs, but that the worst bugs were lack of error messages and awkwardness in laying out diagrams.

**Q6. Other UML modeling tools or diagram drawing tools would have been just as effective in the course as using Umple.** Average = 3.3 (Between Neutral and Agree); median = 3.0 (Neutral). There were several comments that students just lacked enough experience with other tools to answer this question with anything but Neutral.

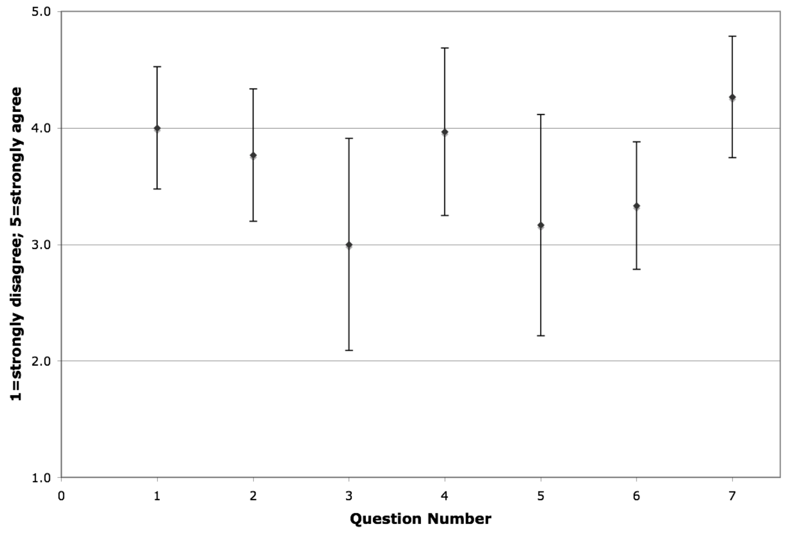

**Figure 2: Mean answers to the first 7 questions in the satisfaction survey, with error bars indicating the standard deviation.**

**Q7. I would like to use Umple in future, assuming that its error messages are improved and its bugs are dealt with.** Average = 4.3 (Above Agree); median = 4.0 (Agree). Nobody disagreed and 30% said Strongly Agree. This result certainly seems to be a ringing endorsement.

The above results are summarized in Figure 2, where error bars indicate standard deviation.

The final three questions were to be answered on a scale with values 'Hate it', 'Dislike it', 'Neutral', 'Like it', and 'Like it a lot', again rated using a 0 to 5 scale for analysis.

**Q8. The availability of a textual form for writing models and creating UML diagrams.** Average = 4.1 (I like it); median = 4.0 (I like it). Nobody disliked it.

**Q9. The ability to edit the diagram and have the Umple text update, and vice versa.** Average = 4.3 (above 'I like it'); median = 4.0 (I like it). Nobody disliked it.

**Q10. The ability to generate code in Java and other languages to represent the UML design.** Average = 4.4 (between 'I like it' and 'I like it a lot); median = 4.5. Nobody disliked it.

The results above are summarized in Figure 3

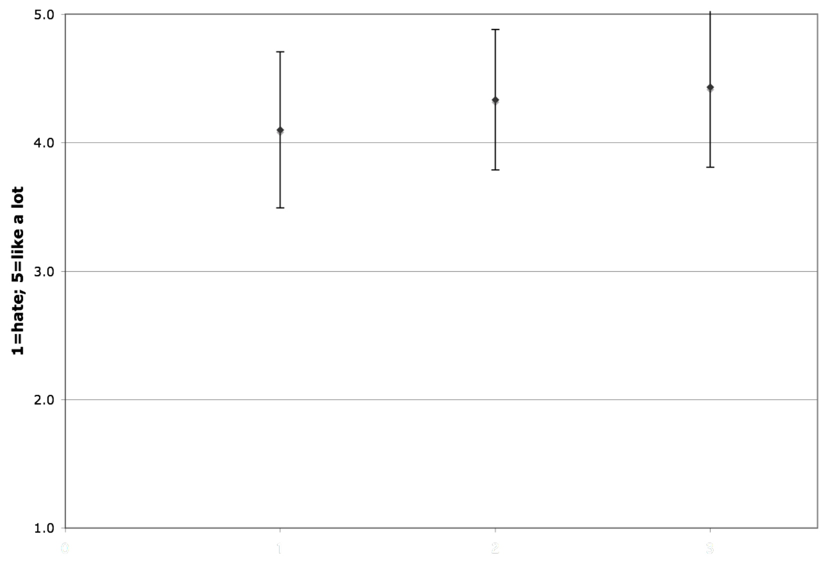

**Figure 3: Mean answers to Q7-9 regarding how much features were liked: the textual form (left); code-diagram duality (centre) and code generation (right). Error bars show standard deviation**

The above results clearly suggest it would be beneficial to continue using Umple, especially if improvements continue to be made. They also suggest that other professors might want to consider adopting Umple or a technology like it.

## 5.1. Threats to validity

Even though the course instructor (the first author) could not tell who gave responses, there is some risk that some students gave positive responses simply to please the professor. Also the sample cannot be considered completely random since about 20% of students in the class have a habit of never participating in lectures, nor other activities unless attendance is taken and counts for the grade. It might be that these students would have had different opinions.

## 6. Comparison of grades

In each iteration of the second-year course there has been a midterm exam in the week after completing the core of the teaching material on class diagrams, and a final exam about 50 days later. In each of these exams, students are asked to create a class diagram giving the system domain model corresponding to a paragraph of requirements. This task is the same task illustrated in Section 1.2. It is performed in class using board work and, more recently, Umple. Students also do this task in an assignment, but that is only due after the midterm exam.

The weight of the class diagram questions varied from 30 to 50 percent on the midterm, and from 10 to 20 percent on the final exam. In the following analysis, the weightings and each student's score has been normalized to be out of 100.

The questions were marked using a consistent marking scheme. 50% of the grade is based on completeness; in other words all the expected elements are present, such as classes associations, generalizations and attributes. Generally, the students' ability to model operations was left to other questions. 10% of the grade is based on good naming of elements present; 20% is based on correct multiplicity, generalization, and other syntax elements such as composition diamonds; and 20% is based on whether the design is in fact logically correct.

**Grades prior to Umple**: In the four times the first author taught this course immediately prior to introducing Umple (2003, 2004[2], 2007, and 2008): The average midterm exam grade on UML class diagram questions was 74.4%. The average final exam grade on such questions was 76.4%.

**Grades since using Umple**: In the two iterations since students have used Umple (2009 and 2010) the average midterm grade on UML class diagram questions was 82.3%. This is a 10.6% improvement. Furthermore, the average final exam grade (2009 only[3]) on such questions was 88.8%. This is a 15.9% improvement. There was no noticeable difference in grades for questions not relating to UML diagrams

We also were able to compare grades for students in the fourth year course: Those students who submitted an Umple solution were marked on average 9.9 percentage points higher for their Umple solution compared to their UML solution in 2009 and on average 14.8 percentage points higher in 2010.

The above results suggest that the use of Umple has indeed helped students learn how to model.

### 6.1. Threats to validity

There are several important threats to validity of these results. Firstly, although the difficulty of the questions was kept as similar as possible, it may be by chance that since the introduction of Umple the questions ended up being naturally easier. Also, the first author did all the grading of exams, and it is possible that he is becoming less critical with age – certainly there is considerable latitude for subjectivity in marking. In addition, the weights given to the questions were not equal in every exam, so students may have put more effort into questions that were weighted higher. As a result of these factors, we suggest that these results should be taken as suggestive evidence, subject to replication. We have not done a

[2] The first author did not teach the course for two years because he was Associate Dean in 2005 and was on sabbatical in 2006.

[3] If the paper is accepted the data will be enhanced to include 2010 final exam data.

formal statistical analysis due to the fact that the threats would render ‘statistical significance’ somewhat moot in our opinion. Such analysis could be done once a greater volume of data is gathered, with some of the threats being better controlled. This is planned as future work.

## 7. Conclusions

Web-based technology for editing UML diagrams live in the classroom, and displaying code generated from models, makes lecturing about modeling and UML easier and faster for the professor. Students also like this technology, which we call Umple. They believe it helps them learn, and were able to use it successfully in their homework assignments, laboratory exercises and projects. In addition we also have noticed improvements in students’ grades following the introduction of the technology.

Opportunities for future work include more formal studies, comparing comprehension in cases where Umple is used and not used.

## References

[1] A. Forward and T.C. Lethbridge, “Problems and Opportunities for Model-Centric Versus Code-Centric Software Development: A Survey of Software Professionals,” in MiSE '08: Proceedings of the 2008 International Workshop on Models in Software Engineering, 2008. DOI=10.1145/1370731.1370738, pp. 27-32.

[2] A. Forward and T.C. Lethbridge, “Perceptions of Software Modeling: A Survey of Software Practitioners”, Technical Report, http://www.site.uottawa.ca/~tcl/gradtheses/aforwardphd/TR-2008-07-Survey-On-Software-Modeling-Forward-Lethbridge.doc

[3] J. Rumbaugh, M. Blaha, W. Lorensen, F. Eddy and W. Premerlani, “Object-Oriented Modeling and Design”, Prentice Hall, 1990.

[4] E. Gamma, R., Helm, R., Johnson, and J., Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. New Jersey: Addison-wesley Reading, MA, 1995.

[5] T.C. Lethbridge, and R. Laganière, Object-Oriented Software Engineering: Practical Software Development using UML and Java. New York, NY, USA: McGraw-Hill, 2005.

[6] T.C. Lethbridge, and R. Laganière, Object-Oriented Software Engineering Powerpoint Slides, http://www.site.uottawa.ca/school/research/lloseng/supportMaterial/slides/

[7] A. Forward, “The Convergence of Modeling and Programming: Facilitating the Representation of Attributes and Associations in the Umple Model-Oriented Programming Language”, PhD. Thesis, University of Ottawa, 2010 http://www.site.uottawa.ca/~tcl/gradtheses/aforwardphd/

[8] A. Forward, T.C. Lethbridge, and D. Brestovansky, (2009), “Improving Program Comprehension by Enhancing Program Constructs: An Analysis of the Umple language”, International Conference on Program Comprehension (ICPC) 2009, Vancouver, IEEE Computer Society, pp. 311-312.

[9] T.C. Lethbridge, A. Forward, and O. Badreddin, (2010), “Umplification: Refactoring to Incrementally Add Abstraction to a Program”, Working Conference on Reverse Engineering, Boston, October 2010, pp. 220-224.

[10] D. Brestovansky, “Exploring Textual Modeling using the Umple Language”, Masters thesis, University of Ottawa, 2008

[11] UmpleOnline, http://cruise.site.uottawa.ca/umpleonline/ Accessed Nov 2010

[12] T.C. Lethbridge, “SEG2105: Introduction to Software Engineering”, http://www.site.uottawa.ca/~tcl/seg2105/, accessed November 2010

[13] ACM and IEEE Computer Society, “Software Engineering 2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering”, http://sites.computer.org/ccse/SE2004Volume.pdf

[14] T.C. Lethbridge, R. LeBlanc, A. Sobel, T. Hilburn, and J. Díaz-Herrera, (2006), “SE 2004: Recommendations for Undergraduate Software Engineering Curricula”, IEEE Software, Nov-Dec 2006, pp. 19-25

[15] T.C. Lethbridge, and G. Musbacher, “SEG4110: Advanced Software Design and Reengineering”, http://www.site.uottawa.ca/~tcl/seg4110/, accessed November 2010

[16] ArgoUML, http://argouml.tigris.org/, accessed November 2010

[17] Rational Software Modeler, http://www-01.ibm.com/software/awdtools/modeler/swmodeler/, accessed November 2010

[18] SEG2105 Lab 4 Instructions, http://www.site.uottawa.ca/~tcl/seg2105/coursenotes/SEG2105Lab4-Oct2010.doc