An Empirical Experiment of Comprehension on Textual and Visual Modeling Approaches

University of Ottawa Computer Science Technical Report TR-2011-03

This document reports on an experiment that takes a human comprehension perspective on approaches to represent the design of software systems. To facilitate replication, this document includes experiment specifications, results, and analysis. The experiment investigates three notations: UML, Java, and Umple [1]. Participants are asked to answer questions that reflect their level of comprehension. The results reveal that for simple comprehension tasks, the pure visual UML notation and a Umple textual language are comparable. Comprehension of Java code implementing a UML/Umple model was lowest of all three notations. Our results align with the intuition that raising the abstraction levels of common object-oriented programming languages enhances comprehensibility, and confirms that textual notations for modeling are just as viable as diagrammatic notations.

We created representations of semantically-equivalent small systems using UML, and Umple and Java. These representations were then presented to participants who were asked to answer straightforward comprehension questions. By measuring the participants' response times, we should be able to infer which notation aided the subjects' comprehension of the system more effectively than the other.

Prior to the commencement of the experiment, participants watch a short video training session on UML and Umple [2, 3]. Participants are assumed to have decent knowledge of Java and require no Java training. After each presentation of an example system in one of the three notations to the participant, the participant is asked 12 questions. The questions are simple and participants' answers are expected to be correct in most cases. Average response time for each question is expected to be less than 30 seconds. In total, each participant will have provided answers to 36 questions (12 questions per example). The experiment sessions with participants are audio recorded. The results are analyzed quantitatively.

Introduction

UML has emerged as the defacto standard for representation of software engineering diagrams. It is widely accepted that using visual modeling, such as UML, gives favorable results in the creation and maintenance of software artifacts. At the same time, there is evidence that visual modeling adoption in the software engineering industry remains low [4]. The open source community for example remains almost entirely code centric.

In recent years, a number of approaches have emerged that use similar visual modeling abstractions but are represented textually. Object Management Group (OMG), the organization that manages the UML standards, has proposed in the past HUNT, a textual notation for UML class diagrams. More recently, Alf [5], a concrete textual syntax for UML action semantics, has been proposed. Certain programming languages, such as Ruby, support some UML modeling abstractions, but no language supports associations and state machines fully.

Umple, one of the notations evaluated in this study, [6]; is a model-oriented language where modeling abstractions such as state machines and associations are embedded in code or vice-versa. Other textual modeling approaches includes MetaUML, yUML, TextUML, State Machine Compiler (SMC), AsmL [7], and Executable UML [8].

There seems to be little evidence, if any, that such approaches are indeed effective, and whether they enhance or hinder comprehension. For example, there are a number of studies that indicated gains in the comprehensibility when using visualizations while others reported that graphics were significantly slower than text in the experimental comprehension tasks [9]. Moreover, what is the level of abstraction that is ideal for comprehension? In other words, what is the level of abstraction above which the marginal utility of additional abstraction becomes negative?

The literature is rich with theoretical analytical work on comprehension of notation. Cognitive Dimensions framework [10], for example, provides valuable perspective on notation and comprehension. This experiment compliments such analytical work with empirical results .

Experiment definition, context, and steps

Participants are presented with hard-copy samples of example systems in UML, Umple, and Java. This aims at eliminating the learning curve associated with modeling tools – for UML models – and text editors – for Java and Umple. The experiment takes place in a controlled environment.

Step 1
Participants are asked pre-experiment profiling questions. Duration: 2 minutes.

Step 2
Participants watch a short video on the notation (UML and Umple). Participants are expected to be familiar with Java and require no Java training. Duration: 6 minutes.

Step 3
Participants are provided with three instances of example systems, one at a time. Participants are given 1 minute to review the systems, and they are not allowed to ask any questions. After each presentation, the participants are asked to answer a number of questions. Duration: 20 minutes.

Experiment Goals

The goal of this experiment is to evaluate the Umple textual modeling notation in comparison with UML and Java. One of the objectives behind the development of Umple was that it should gain the advantages of both visual modeling (UML) and textual programming (i.e. Java). This experiment, therefore seeks to validate the hypothesis that Umple has retained the advantages of UML with respect to comprehension. We leave it as separate research to assess whether Umple has also retained any advantages of textual programming.

A sub-goal of this experiment is to evaluate the effect on experimental results of using software artifacts named with identifiers derived from the domain versus abstract names.

Experiment Metrics

The following are the experiment metrics.

1. Number of questions answered correctly and incorrectly
2. For each question:
   2.1 Time elapsed for the first answer
   2.2 Time elapsed for a correct answer
   2.3 Number of incorrect answers (i.e number of trials)
3. Pre-experiment profiling questions

Problem Statement

The experiment employs three hypotheses:.

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.

The next hypothesis is similar, comparing Umple and UML diagrams:

H2: A system written in Umple is more comprehensible than an equivalent UML diagram of the system.

H2o: Umple and UML diagrams do not differ in comprehensibility

The third hypothesis was of secondary interest:

H3: Whether names derived from the domain or abstract names are used has an effect on experiment results regarding comprehensibility.

H3o: The use of abstract or domain-derived names makes no difference to comprehensibility.

Using systems with domain names or abstract names may or may not have an effect on response time and number of inaccurate responses. The purpose of testing H3 is to determine whether, in future experiments, we need to care about controlling for this factor.

Experiment Planning

The experiment recruited 9 participants. Each participant is presented with three different example systems (one, two, and three in

Experiment Objects

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). Three of the models used names derived from the domain (student-supervisor domain), while the remaining six models used abstract names (i.e. a, b, c). Abstract names were used since we wanted to test the 'pure' comprehensibility of the notations, and wanted to avoid the threat to validity that people might understand the system simply because they understand the underlying domain. On the other hand we also used names derived from the domain to reduce the opposite threat to validity, which is that systems with abstract names are less realistic.

Prior to use in the experiment the example systems and the renderings of the systems in each notation were reviewed by three independent researchers to help maintain consistent complexity levels across the modeling examples. The following two tables provide summaries of the number of modeling elements in each example system.

Table 3) in three different notations (Umple, UML, and Java).

Table 1: example instances distribution

Instance/Participants distribution Notation	Umple	UML	Java
Participant 1	One	Three	Two
Participant 2	Two	One	Three
Participant 3	Three	Two	One
Participant 4	One	Three	Two
Participant 5	Two	One	Three
Participant 6	Three	Two	One
Participant 7	One	Three	Two
Participant 8	Two	One	Three
Participant 9	Three	Two	One

Three of the models used names derived from the domain (student-supervisor domain), while the remaining six models used abstract names (i.e. a, b, c). Abstract names were used since we wanted to test the 'pure' comprehensibility of the notations, and wanted to avoid the threat to validity that people might understand the system simply because they understand the underlying domain. On the other hand, we also used names derived from the domain to reduce the opposite threat to validity, which is that systems with abstract names are less realistic.

Table 2: Domain and abstract naming distribution

Distribution of Models with Domain and Abstract names	Umple	UML	Java
Participant 1	DOMAIN	abstract	abstract
Participant 2	abstract	DOMAIN	abstract
Participant 3	abstract	abstract	DOMAIN
Participant 4	DOMAIN	abstract	abstract
Participant 5	abstract	DOMAIN	abstract
Participant 6	abstract	abstract	DOMAIN
Participant 7	DOMAIN	abstract	abstract
Participant 8	abstract	DOMAIN	abstract
Participant 9	abstract	abstract	DOMAIN

Design Validation – Pilot Study

In order to initially verify and validate the design of the experiment, as well as identify potential flaws in the design, we conducted a pilot study. The pilot study was done with three participants, who were selected based on convenience and software engineering background.

From the pilot study it was found that some of the question wording was not clear. It was also found that participants tend to get bored by the end of the experiment. The question wording was corrected

and reviewed independently by three researchers. The boredom was mitigated by reducing the number of questions, and giving participants a 2 minute break between rounds.

Experiment Objects

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). Three of the models used names derived from the domain (student-supervisor domain), while the remaining six models used abstract names (i.e. a, b, c). Abstract names were used since we wanted to test the 'pure' comprehensibility of the notations, and wanted to avoid the threat to validity that people might understand the system simply because they understand the underlying domain. On the other hand we also used names derived from the domain to reduce the opposite threat to validity, which is that systems with abstract names are less realistic.

Prior to use in the experiment the example systems and the renderings of the systems in each notation were reviewed by three independent researchers to help maintain consistent complexity levels across the modeling examples. The following two tables provide summaries of the number of modeling elements in each example system.

Table 3: Example model properties

Number of modeling elements								
	Classes	**Assoc.**	**Attr.**	**States**	**Trans**	**Guards**	**events**	**Actions**
Example One	3	3	3	4	4	1	3	1
Example Two	3	3	3	4	4	1	3	1
Example Three	3	2	1	3	5	1	5	5

Note: Each example is represented in three notations; Umple, UML, and Java

Question List

There are 12 questions per example system. The questions wording and correct answers have nine variations. This is because a question posed on a Java implementation must use different wording than a question posed on a UML model. The questions for UML and Umple are almost identical.

The nine question variations and their model answers are listed in the

Appendix.

Information Collection Prior to Experiment

Table 4 lists the questions posed at the onset of the experiment.

Table 4: Information collected prior to the experiment

Education
Are you a bachelor student? Masters? PhD? What year? Which university? How many Software Engineering courses have you successfully completed?
Background *
How familiar are you with Java? How familiar are you with UML? How familiar are you with Umple?
Experience
How many years/months of working experience in the software development industry?
* **Note:** This scale is used for these questions (Never heard of it, know about it but not used it, used it a little, Have used it regularly, expert)

Selection of Participants

The nine participants are selected randomly from the pool of university students and professionals who have completed at least two courses in OO-programming. Participation in the experiment is anonymous and voluntary. Participants are not compensated for their participation.

Variables in the Study

The experiment employs the following variables.

Extraneous Variables

These variables may have an effect on the dependent variables. The experiment design attempts to eliminate or minimize the effect of extraneous variables.

Table 5: Extraneous variables

Domain knowledge
Participants' knowledge of the domain should have no impact on their responses. The experiment example systems are very simple and

domain knowledge should not have an impact on answers.

Model complexity levels

All example instances used in the experiment have identical complexity levels, as possible. This is ensured by using the same number of model properties across example systems (i.e. number of classes, associations, attributes, states, transitions, actions, guards, and events)

Java, Umple, and UML background and experience

Participants are expected to have good knowledge in Java. That is because the pool of potential participants is drawn from university students who have completed at least two courses in OO programming. Knowledge of UML and Umple are variable. The following is performed in order to isolate the impact such knowledge and background on results:

1. Training examples are presented to the participants prior to commencing the experiment.
2. The examples used are very simple, requiring only very basic knowledge of the different notations (Java, Umple, UML).

Learning during the experiment

Participants will inevitably learn aspects about the model instances during the experiment. The following is performed in order to isolate the impact of learning on results:

1. Example systems are slightly different. Participants cannot answer questions about the first instance based on their knowledge of the first instance.
2. Every model instance is presented in a different notation.

Environment factors

Every effort is taken to eliminate or minimize external environmental factors, such as noise and interruptions.

Independent Variables

These are the variables that are manipulated during the experiment.

Table 6: Independent variables

Notation
The notation used to present the system examples. This variable is manipulated (UML, Umple, Java).
Domain Names and Abstract Names
Model element naming are manipulated (Domain names versus abstract names).

Dependent Variables

These are the responses observed during the experiment. The independent variables manipulations should ideally be solely responsible for the variations in the dependent variables.

Table 7: dependent variables

Time to respond to questions
The time the participant takes to determine the answer to the question
Number of correct responses
The number of correct responses on the first trial of answering each question

Threats of Validity

Threats of validity of this experiment are summarized in Table 8.

Table 8: Threats to validity

Expertise and background of participants may affect how fast they respond to questions
Threat *Participants may have varying background and experience with modeling using UML and Umple. Their background may affect how fast they are able to respond to questions.*
Mitigation 1. We collect demographic information on the users and make sure that distribution of background and experience is balanced. 2. At the beginning of the experiment, we present a short tutorial that covers knowledge required for answering the experiment questions. 3. Experiment examples are very simple and require minimal expertise. 4. The experiment is balanced, participants with high levels of expertise will most likely answer better and quicker answers in all three instances of the model.

Number of example systems and participants may not be representative of the overall population

Threat

This is an external validity threat. Can we generalize to the results in this study to the general population? This threat is strengthens by use of student participants.

This validity threat is existent in any controlled experiment.

Mitigation

1. *Using three example systems, rather than one.*
2. Randomly recruiting nine participants.
3. Effort to recruit some professional participants.
4. Making experiment design, procedure, and models public to invite replication.

Question and example variations have an impact on response time

Threat

To accommodate the three notations under study (Java, Umple, UML) there are three slightly different system examples, and 9 question lists. The question essence is the same, but the wording is different. There is a threat that these variations affect the response time of participants.

Mitigation

1. The questions and example systems have been reviewed by at least 3 experts independently to make sure that the variations have no, or little, impact on response time.
2. Conducting at least one prototype for each question set to make sure the questions are unambiguously understood by participants.

Results[1]

A total of nine participants provided answers to 36 questions; twelve questions per example for three examples. In total, there are 324 response times recorded.

Seven participants reported to have a PhD or enrolled in a PhD program. Two participants reported being a Master's student. The majority came from University of Ottawa (seven). There was one student from Carleton and one from Cornel University.

Participants had an average of four software engineering courses. They had an average Java familiarity score of 3.3, average UML familiarity of 2.7, and an average Umple familiarity of 1.67. Participants had an average of 9.9 months of professional software development experience.

Table 9 summarizes the average response time for the nine participants. For example, participant number 1 responded to Umple questions in 5.5 seconds on average, and 4.7 to UML questions on average, and 9.3 seconds to Java questions on average.

The table also shows the average per notation. For example, Umple's overall response time is 3.57, compared with 3.61 for UML, and 6.87 for Java.

[1] The raw data is included in the attached excel sheet (Results.xls)

Also in the table is the average per participant. For example, participant number 1 response time was 6.5 on average across the three different notation types.

Table 9: Average results

Participant	Umple	UML	Java	Overall average
1	5.5	4.7	9.3	**6.5**
2	5.0	4.2	9.0	**6.1**
3	3.7	3.6	6.2	**4.5**
4	2.3	3.1	6.2	**3.9**
5	2.1	3.0	6.3	**3.8**
6	2.2	2.4	3.4	**2.7**
7	4.3	2.4	6.0	**4.2**
8	3.0	4.0	6.3	**4.4**
9	4.1	5.1	9.0	**6.1**
Average	**3.57**	**3.61**	**6.87**	

It is our intention that the questions be straightforward and participants should be able to provide correct answer at the first attempt. However, it was not always the case; there were a total of 37 incorrect responses out of the 324 questions posed. Incorrect responses were distributed among all three notations as follows: Umple with 8, UML 12, and Java 17.The average response time per example is summarized in Table 10; to two significant figures they are identical.

Table 10: Average response time per example

Average Per Example	
E1	4.67
E2	4.74
E3	4.67

Questions posed about example systems that use domain names had an average response time of 4.67 seconds. On the other hand, models that used abstract names had an average response time of 4.79 seconds.

Standard Deviation

Table 11 summarizes the standard deviation for the experiment results.

Table 11: Standard deviation per participant

	S.D per participant		
	Umple	**UML**	**Java**
1	7.2	5.6	13.4

2	9.4	6.5	13.2
3	4.7	3.5	9.5
4	2.5	3.4	9.0
5	1.8	4.3	9.5
6	1.9	1.8	3.2
7	10.3	1.8	9.2
8	2.7	3.9	8.1
9	4.6	5.6	11.4
Average	**5.0**	**4.0**	**9.6**

Table 12 summarizes the profiling information results.

Table 12: Profiling information

Profiling Info			
	PhD	**Masters**	**Other**
Educational Background	6	2	1
University	Ottawa (6)	Carleton (2)	Cornel (1)
Average software engineering courses	4.0		
Average Java Familiarity	3.3		
Average UML Familiarity	2.7		
Average Umple familiarity	1.67		
Average months of S/W dev experience	9.89		

Results Analysis

This section presents our analysis and interpretation of the experiment data and is organized as follows. We first re-examine the experiment threats of validity and make an assessment of the identified threats. We then present the results of the parametric and non-parametric statistical analysis and results.

Assessment of threats of validity

Profiling information indicate that participants have more Java background than UML, and much less background on Umple. Therefore, if background were to have an impact, it will not be in favor of Umple.

The results of the average response time per example (Table 10) reveals that example do not seem to have an impact on the response time of the participants.

Examining Data for Umple and Java

Figure 1 illustrates the average response time for Umple and Java.

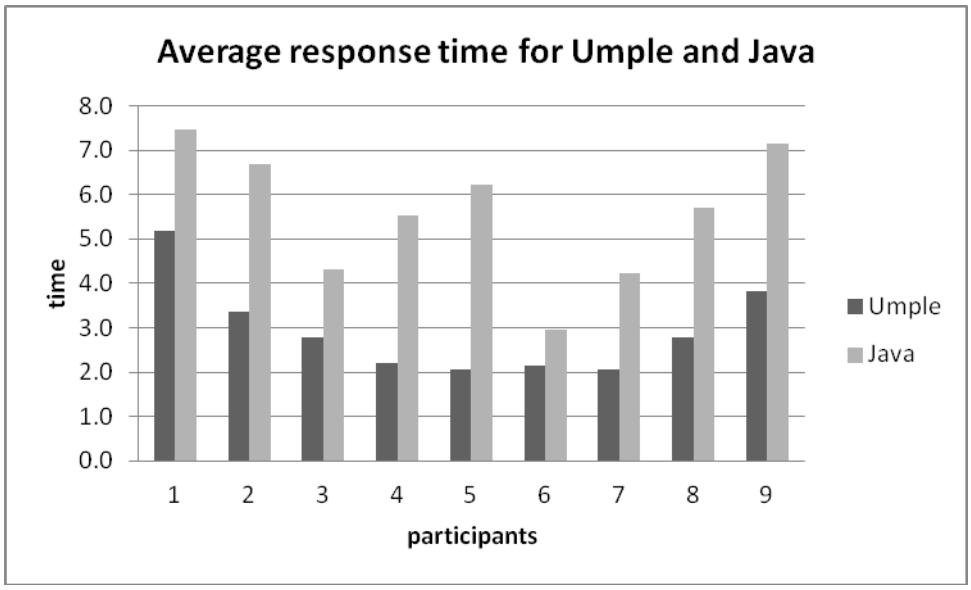


Figure 1: Average response time for Umple and Java

Using a two-tailed t-test to measure the statistical significance, Umple is better than Java ($p=1.5x^{10-8}$).

Using Mann-Whitney test (U-test) Umple is better than Java ($p = 8.906x^{-09}$) and a W value of 2722.5.

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 ($P=6.0103x^{-14}$).

Examining data for Umple and UML

Figure 2 illustrates the average response time for Umple and UML.

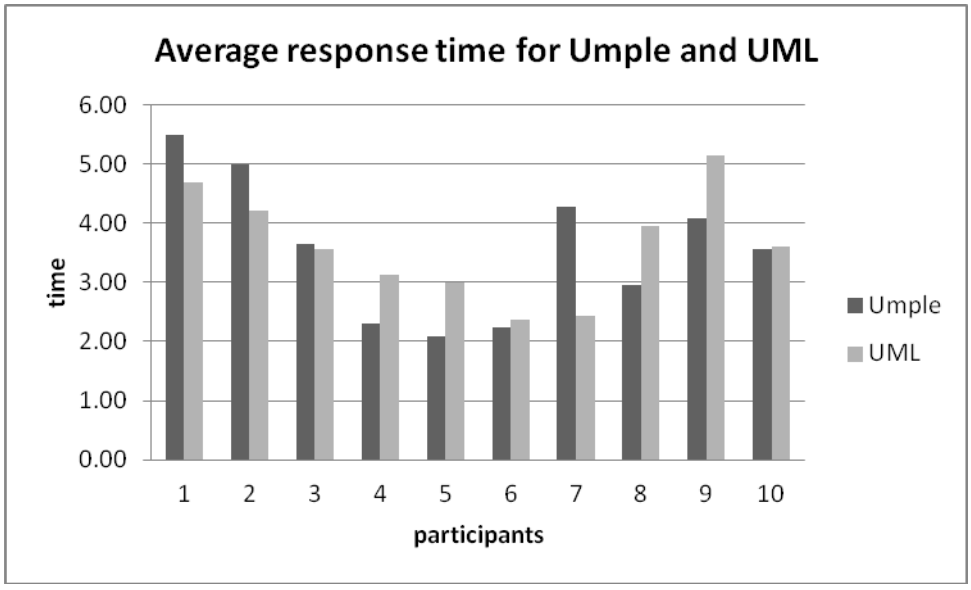


Figure 2: Average response time for Umple and UML

Using a two-tailed t-test to measure the statistical significance, Umple is not significantly better than UML (P=0.9).

Using the 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 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 [11]. 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 the different populations. Elsewhere in the literature, this technique is also used to identify outliers [12].

Regarding the third hypothesis (H3), using the Mann-Whitney test (U-test), we cannot conclude that using domain or abstract names can have any significant impact of the average response time (W= 3, and P =0.6). We come to the same conclusion by using the standard deviation analysis described earlier.

Discussion

There is evidence that Umple performed significantly better than Java. But such evident is lacking in the case of UML. Our interpretation of the results is as follows.

The tasks involved in this experiment focused on simple model comprehension and tracing questions. These tasks resemble realistic software engineering tasks [13]. These tasks, however, do not cover the wide spectrum of tasks performed by software engineers. In particular, the tasks do not address model creation, tuning, implementation, and maintenance tasks. Therefore, interpretation of the results must take into consideration the scope on which conclusion can be drawn.

We can therefore infer that Umple is better than Java in understanding a system. We can also infer that Umple is not significantly better than UML in this regard. We can with significant confidence claim that Umple is not worse than the visual UML models. After all, Umple is not meant to replace UML, but to complement it. Indeed, the UmpleOnline tool [6] allows both to be used interchangeably.

A core lesson from this technical report is that people whose program development approach is primarily textual, for any reason, should with confidence consider Umple as a viable textual technology. It retains the advantages of text, while being easier to understand than Java, and being just as comprehensible as UML diagrams when it comes to UML concepts such as state machines and associations

Future work

This experiment cannot be a final word on model notation effectiveness, and it is not intended to be so. Future work to replicate this experiment can be of great value in two ways. First, by increasing the number of participants; second, by recruiting more professional software engineers and make conclusions on this group of subjects; and third, by using a variety of more complex systems.

It is yet to be seen in future studies how Umple, UML, and Java compare in the performance of other, possibly more elaborate, software engineering tasks. One variant of this experiment can ask participants to spot flaws or defects in model elements, or match pieces of Umple models and Java artifacts to UML models. Such tasks can shed more light on the nature of comprehension of textual modeling.

References

[1] 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.

[2] Youtube video. Accessed 2011. http://www.youtube.com/watch?v=HIBo0ErCVtU.

[3] Youtube video. Accessed 2011. http://www. Youtube. com/watch?v=mFczzVkTZ9g.

[4] Forward, A. and Lethbridge, T. C. "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, pp. 27-32.

[5] "Concrete Syntax for a UML Action Language, Action Language for Foundational UML," accessed 2010, http://lib.modeldriven.org/MDLibrary/trunk/Applications/Alf-Reference-Implementation/doc/

[6] "UmpleOnline," accessed 2011, http://cruise.site.uottawa.ca/umpleonline/

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

[8] "Model Driven Architecture with Executable UML (TM)". Cambridge University Press New York, NY, USA, 2004,

[9] D. Hendrix, J. H. Cross II and S. Maghsoodloo. "The effectiveness of control structure diagrams in source code comprehension activities". 2002. IEEE Trans.Software Eng.pp. 463-477.

[10] T. R. G. Green. "The cognitive dimension of viscosity: a sticky problem for HCI". 1990. Hum.-Comput.Interact.pp. 79-86.

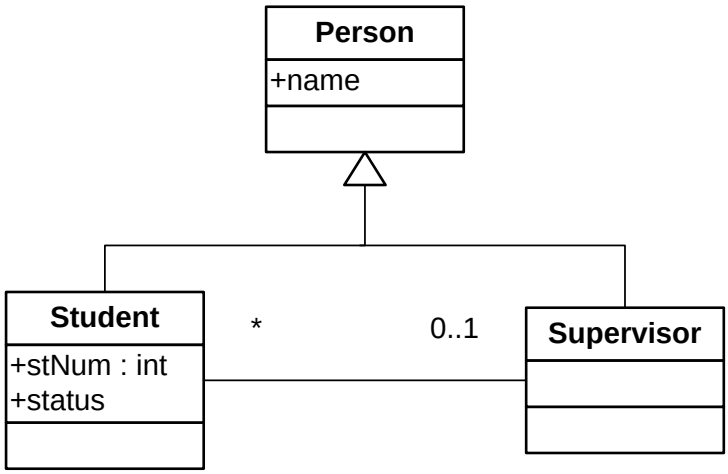
[11] S. Mohammad. "From once upon a time to happily ever after: Tracking emotions in novels and fairy tales". 2011. ACL HLT 2011pp. 105.

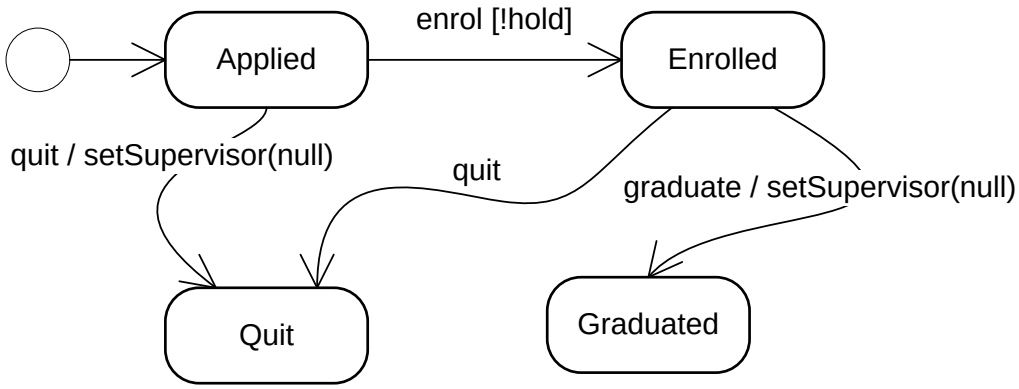
[12] S. M. Mohammad and P. D. Turney. "Crowd-sourcing a word--emotion association lexicon". 2011. In submission.

[13] Sjoberg, D. I. K., Anda, B., Arisholm, E., Dyba, T., Jorgensen, M., Karahasanovic, A., Koren, E. F. and Vokác, M. "Conducting realistic experiments in software engineering," in *Empirical Software Engineering, 2002. Proceedings. 2002 International Symposium n,* 2002, pp. 17-26.

Appendix

Example System One (UML)





Example System One (Umple)

Umple

```
class Person {
  name;
}

class Student {
  isA Person;

  Integer stNum;
```

```
  status {
    Applied {
      quit -> Quit;
      enrol [hold] -> Enrolled;
    }
    Enrolled {
      quit -> /{setSupervisor(null);}
        Quit;
      graduate -> /{setSupervisor(null);} Graduated;
    }
    Graduated {}
    Quit {}
  }

  * -- 0..1 Supervisor;
}

class Supervisor {
  isA Person;
}
```

Example System One (JAVA)

Student

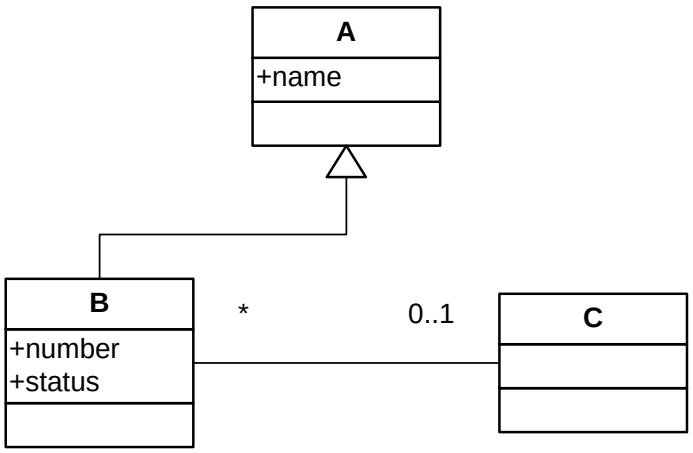
```
class Student extends Person {
   private int stNum;

   boolean hold;
   private int status;
   private Supervisor mySupervisor;

   public Student(int stNum) {
     this.stNum= stNum;
     status=0;
   }

   public int stNum() {return stNum;}

   public void enrol() {
     if (!hold){
       if(status ==0) status=1;}
  }
   public void graduate() {
      if(status==1) {
         removeSupervisor();
         status=2;
      }
   }

   public void quit() {removeSupervisor(); status=3;}
```

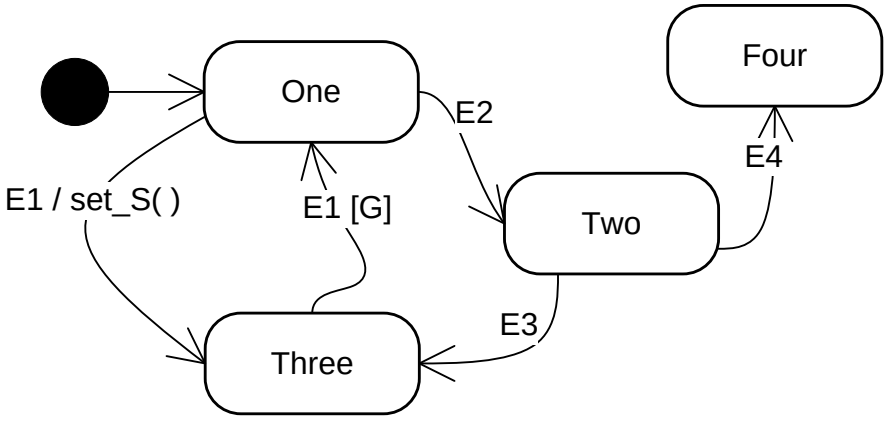
```
    public boolean setSupervisor(Supervisor newSupervisor) { }
    public boolean removeSupervisor() { }
 }
```

Person	Supervisor

class Person {
  public String name;
}
	class Supervisor extends Person {
	  List<Student> mentees = new
	ArrayList<Student>();
	
	  Supervisor() {}
	}

Example System Two (UML)





Example System Two (Umple)

Umple

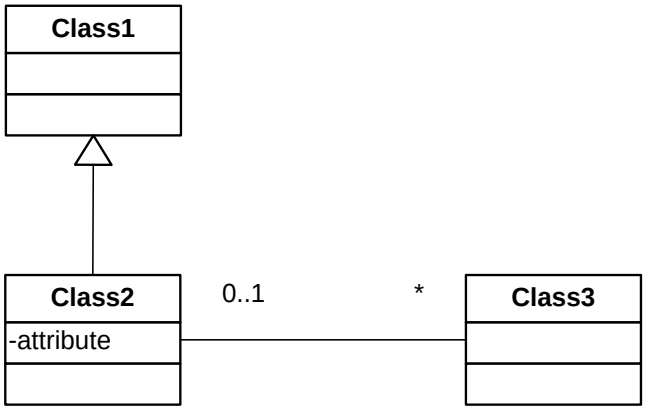
```
class A {
  name;
}

class B {
  isA A;
  * -- 0..1 C;

  Integer number;

  status {
    One {
      E1 -> /{set_S();} Three;
      E2 -> Two;
    }
    Two {
      E4 -> Four;
    }
    Three {
      E1 [G] -> One;
    }
    Four { }
  }
}

class C { }
```

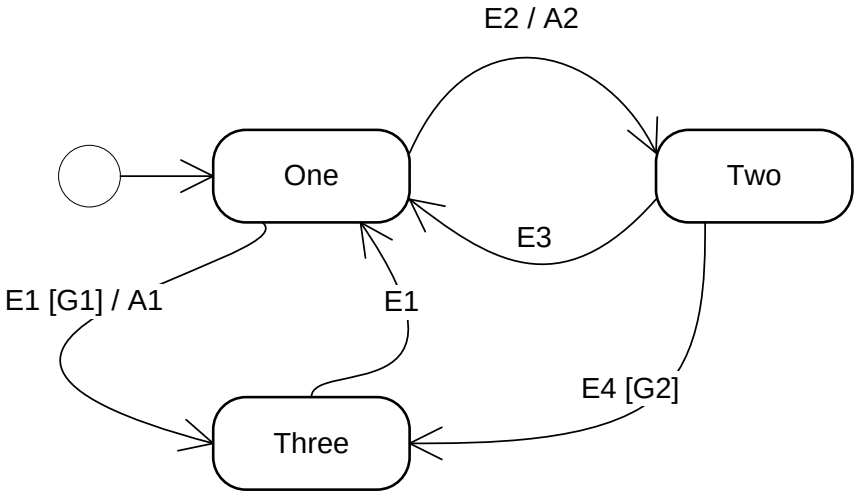
Example System Two (JAVA)

Class B

```
class B extends A {
   public int number;

   private int status;
   private C myC;

   public B(int number) {
     this.number = number;
     status = 1;
   }

   public int number() {return number;}

   public void E1() {
     if(status == 1) {
       set_S();
       status = 3;
     }
     if(status == 3 {
       if (G)  {privateInt_2 = 3;}
     }
  }
   public void E2() {
     if(status == 1) {status = 2;}
   }

   public void E3() {
     if (status == 2) status = 3;
   }

   public void E4() {
     if (status == 2) status = 4;
   }

   public boolean setC(C newC) { }
   public boolean removeC() { }
 }
```

Example System Three (UML)





Example System Three (Umple)

Umple

```
class class1 {
}

class class2 {
  isA Person;

  attribute;

  stateMachine {
    StateOne {
      E1 -> / {A1} StateThree;
      E2 -> / {A2} StateTwo;
    }
```

```
    StateTwo {
      E3 -> / {A3} StateOne;
      E4 -> / {A4} StateThree;
    }
    StateThree {
      E5 -> / {A5} StateOne;
    }
  }

  * -- 0..1 Supervisor;
}

class class 3 {
  isA class1;
}
```

Example System Three (JAVA)

Class2

```
class Class2 extends Class1 {

   private int attribute;
      List<Class3> role = new ArrayList<class3>();

   public Class2(int attribute) {
     this.attribute = 0;
   }

   public void E1() {
     A1();
     if(attribute ==1) attribute =3; }

   public void E2() {
     A2();
     if(attribute ==1) attribute =2; }

   public void E3() {
     A1();
     if(attribute ==2) attribute =1; }

   public void E4() {
     A1();
     if(attribute ==2) attribute =3; }

   public void E5() {
     A1();
     if(attribute ==3) attribute =1; }
     }
   }

   public void A1(){  }
```

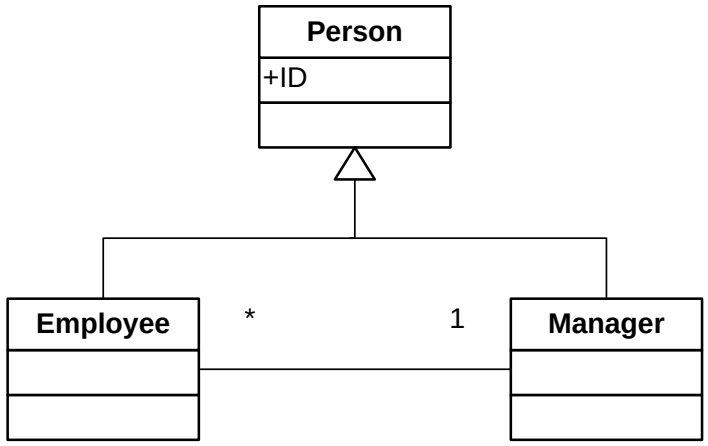
```
   public void A2(){  }
   public void A3(){  }
   public void A4(){  }
   public void A5(){  }
}
```

Class1	Class3

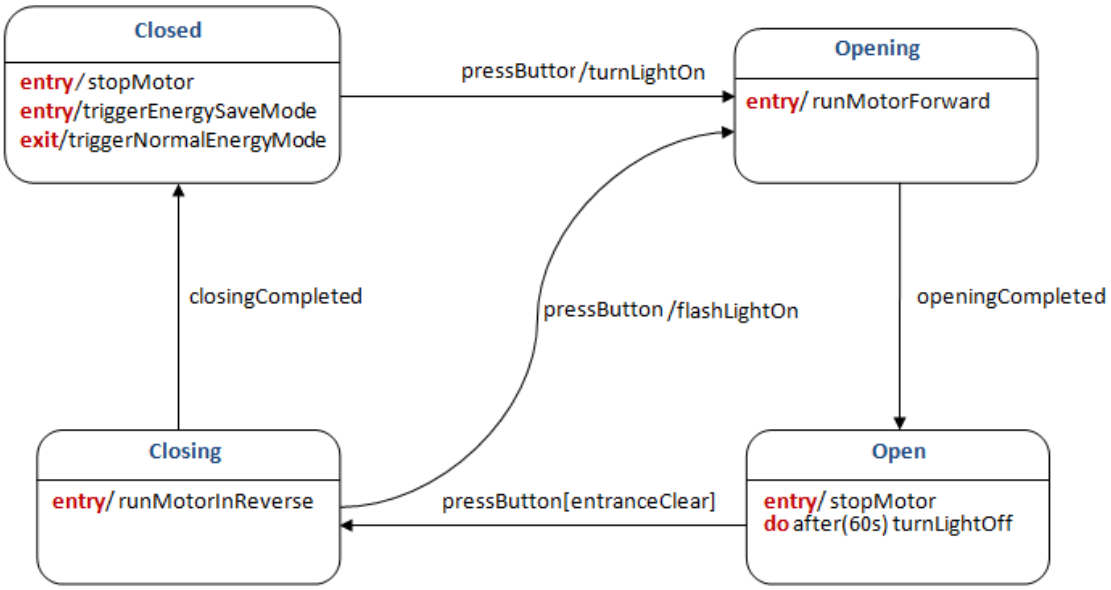
`class Class1 { }`	`import java.util.*;`
	
	`class Class3 extends Class1 {`
	`  private Class2 myClass2;`
	`  Class3() {}`
	`}`

Training Example One (Classes, attributes, Associations)

UML	Umple

	`class Person {`
	`  ID;`
	`}`
	
	`class Manager {`
	`  isA Person;`
	`  1 -- * Employee;`
	`}`
	
	`class Employee {`
	`  isA Person;`
	`}`

Training Example 2 (State Machines)



Question list for example system one

Table 13: Question list for version E1 (UML and Umple)

Questions for Version E1		
	Umple and UML questions	**Correct Answer**
Q1	Let's assume the state machine is in the Applied state and hold is false. Also assume the following events occurred in sequence, enrol, quit, enrol. What is the resulting state?	Quit.
Q2	Assume the student has one supervisor. Can you add another supervisor to the same student?	No.
Q3	Assume a supervisor has 6 students. Can we add another student to this supervisor?	Yes.
Q4	Assume the state machine is in the Applied state, and the value of hold is true. What happens when the event enrol occurs?	Nothing. Not transition occurs.
Q5	How many students can a supervisor have?	Many. Unlimited number.
Q6	What are the possible states the state machine status can have?	Applied, Enrolled, Graduated, and Quit. (in any order)
Q7	What actions are called when the following transition occurs :	Nothing. No actions are

	From Applied to Enrolled	called.
Q8	Can the state machine go directly from Quit to Enrolled?	No.
Q9	Can the state machine go from Graduated to Applied?	No.
Q10	Assume we are in the Applied state, what happens when the event graduate occurs?	Nothing.
Q11	Can you create a Person Object?	No.
Q12	Assume the state machine is in the Applied state and that hold is false. Also assume the following events occur in sequence: graduate, quit, quit, enrol. What is the resulting state?	Quit.

Table 14: Question list for version E1 (Java)

	Questions for Version E1	
	Java questions	**Correct Answer**
Q1	Let's assume the variable status equals zero, and the following functions are called in sequence, enrol(), quit(), enrol(). What is the value of the variable status?	Two.
Q2	Assume the student has one supervisor. Can you add another supervisor to the same student?	No.
Q3	Assume a supervisor has 6 students. Can we add another student to this supervisor?	Yes.
Q4	Assume the value of status equals zero, and the value of hold is true. What happens when the function enrol() is called?	Nothing.
Q5	How many students can a supervisor have?	Many. Unlimited number.
Q6	What are the possible values the variable status can have?	Zero, One, Two, Three, and Four. (in any order)
Q7	What functions are called when the value of the variable status goes from zero to one?	Nothing.
Q8	Can the value of the variable status go directly from two to one?	No.

Q9	Can the value of the variable status go from three to zero?	No.
Q10	Assume the value of status equals zero, what happens when the function graduate() is called?	Nothing.
Q11	Can you create a Person Object?	No.
Q12	Assume the value of status equals to zero. Also assume the following functions are called in sequence: graduate(), quit(), quit(), enrol(). What is the resulting value of the variable status?	Two.

Question list for example system two

Table 15: Question list for version E2 (UML and Umple)

	Questions for Version E1	
	Umple and UML questions	**Correct Answer**
Q1	Let's assume the state machine is "One" state and G is false. Also assume the following events occurred in sequence, E1, E1, E2. What is the resulting state?	Three.
Q2	Assume B has one instance of C. Can you add another instance of C to B?	No.
Q3	Assume C has 6 instances of B. Can we add another instance of B to C?	Yes.
Q4	Assume the state machine is in the "One" state, and the value of G is true. What happens when E3 occurs?	Nothing. Not transition occurs.
Q5	How many instances of B can be associated with C?	Many. Unlimited number.
Q6	What are the possible states the state machine can have?	One, Two, Three, Four. (in any order)
Q7	What actions are called when the following transition occurs : From One to Three.	Set_S()
Q8	Can the state machine go directly from One to Four?	No.
Q9	Can the state machine go from Two to One?	No.
Q10	Assume we are in the One state, what happens when the event E3 occurs?	Nothing.
Q11	Can you create an A Object?	No.
Q12	Assume the state machine is in the One state. Also assume the following events occur in sequence: E2, E3, E3, E1. What is the resulting state?	One.

Table 16: Question list for version E2 (Java)

	Questions for Version E1	
	Umple and UML questions	**Correct Answer**
Q1	Let's assume the variable status equals to 1, and G is zero. Also assume the following functions are called in sequence, E1(), E1(), E2(). What is the value of the variable status?	Three.
Q2	Assume B has one instance of C. Can you add another instance of C to B?	No.
Q3	Assume C has 6 instances of B. Can we add another instance of B to C?	Yes.
Q4	Assume the status variable equals 1, and the value of G is 1. What happens when the function E3() is called?	Nothing. Not transition occurs.
Q5	How many instances of B can be associated with C?	Many. Unlimited number.
Q6	What are the possible values the variable status can have?	1, 2, 3, 4. (in any order)
Q7	When the variable status's value changes from 1 to 3, what function gets called?	Set_S()
Q8	Can the variable status value change directly from 1 to 4?	No.
Q9	Can the variable status's value change from 2 to 1?	No.
Q10	Assume the value of status is 1, what happens when the function E3() is called?	Nothing.
Q11	Can you create an A Object?	No.
Q12	Assume the value of the variable status is 1, and G is false. Also assume the following functions are called in sequence: E2(), E3(), E3(), E1(). What is the resulting value of the variable status?	1.

Question list for example system three

Table 17: Question list for version E3 (UML and Umple)

	Questions for Version E1	
	Umple and UML questions	**Correct Answer**
Q1	Let's assume the state machine is in the One state and G1 is false. Also assume the following events occurred in sequence, E1, E1, E1. What is the resulting state?	One.
Q2	Assume the Class3 has one Class2. Can you add another Class2 to the same Class3?	No.
Q3	Assume a Class2 has 6 instances of Class3. Can we add another Class3 to this Class2?	Yes.
Q4	Assume the state machine is in the One state, and the value of G2 is true. What happens when the event E4 occurs?	Nothing. No transition occurs.
Q5	How many Class3 can a Class2 have?	Many. Unlimited number.
Q6	What are the possible states the state machine can have?	One, Two, and Three. (in any order)
Q7	What actions are called when the following transition occurs : From Two to One	Nothing. No actions are called.
Q8	Can the state machine go directly from Three to Two?	No.
Q9	Can the state machine go from Two to Three?	Yes.
Q10	Assume we are in the One state, what happens when the event E3 occurs?	Nothing.
Q11	Can you create a Class1 Object?	No.
Q12	Assume the state machine is in the One state. Also assume the following events occur in sequence: E2, E3, E3, E2. What is the resulting state?	Two.

Table 18: Question list for version E3 (Java)

	Questions for Version E1	
	Umple and UML questions	**Correct Answer**
Q1	Let's assume the attribute status equals to 1. Also assume the following functions are called in sequence, E1(), E1(), E1(). What is the resulting value of status?	Three.
Q2	Assume the Class3 has one Class2. Can you add another Class2 to the same Class3?	No.
Q3	Assume a Class2 has 6 instances of Class3. Can we add another Class3 to this Class2?	Yes.
Q4	Assume the value of the variable status equals to 1. What happens when the function E4() is called?	Nothing. No transition occurs.
Q5	How many Class3 can a Class2 have?	Many. Unlimited number.
Q6	What are the possible values the status variable can have?	One, Two, and Three. (in any order) (also possible zero since the code initialize to zero.)
Q7	What methods are called when the value of the variable status changes from 2 to 1?	Nothing. No functions are called.
Q8	Can the value of te attribute status changes from Three to Two?	No.
Q9	Can the value of the variable status change from 3 to 1?	Yes.
Q10	Assume variable status is equal to 1, what happens when the function E3() is called?	A1.
Q11	Can you create a Class1 Object? (be flexible here)	No.
Q12	Assume the value of the attribute status equals to 1. Also assume the following methods are called in sequence: E2(), E3(), E3(), E2(). What is the resulting value of the variable status?	Two.