Introducing Software Engineering to General Engineering Students

Mike Rowe, Ph.D.
Department of Software Engineering and Computer Science
University of Wisconsin-Platteville
rowemi@uwpattt.edu

The Introduction to Engineering Projects Course

At the University of Wisconsin-Platteville, an Introduction to Engineering Projects is a required one semester credit course that is typically taken by engineering students in their first year. The course is broken down into seven, two week modules that cover each of the engineering disciplines, Civil, Electrical, Environmental, Engineering Physics, Industrial, Mechanical, and Software Engineering, that are taught at our school. Each week the students meet for a two hour lab and every two weeks the semester the students rotate to another engineering lab. Thus, in one semester they obtain some exposure to each of our supported disciplines. The goals of this course is to provide all engineering students some exposure to other engineering disciplines and provide experience and information to students who have not yet decided on their engineering major.

The last two years, this course has also been offered in a one week condensed version during the summer to high school students who are interested in pursuing engineering majors. The summer school version has been very popular and has grown from two sections the first year to four sections plus a long waiting list this past summer. The high school students optionally earn one college credit if the complete all course requirements including a final exam. The four objectives of the summer version are recruiting students to our university, providing high school students some exposure to engineering disciplines to help them decide which disciplines they might want to pursue, exposing engineering to groups that are traditionally underrepresented in engineering programs, and exposing high school students to a flavor and pace difference between high school and college courses.

The Software Engineering Module

This paper describes the Software Engineering module, which as described above is just one of the seven modules in the Introduction to Engineering Projects course. Knowledge of computing and software engineering is important to all engineering disciplines as well as other STEM students. The US Bureau of Labor Statistics reports that 71% of all new STEM jobs will be computing related[5]. In addition the author believes that developing software is a good experience for any student engineer, in that it can help develop mature process development skills that are useful for all engineers. The problem is that it is difficult to teach students to program, let alone develop a project, in just two weeks.

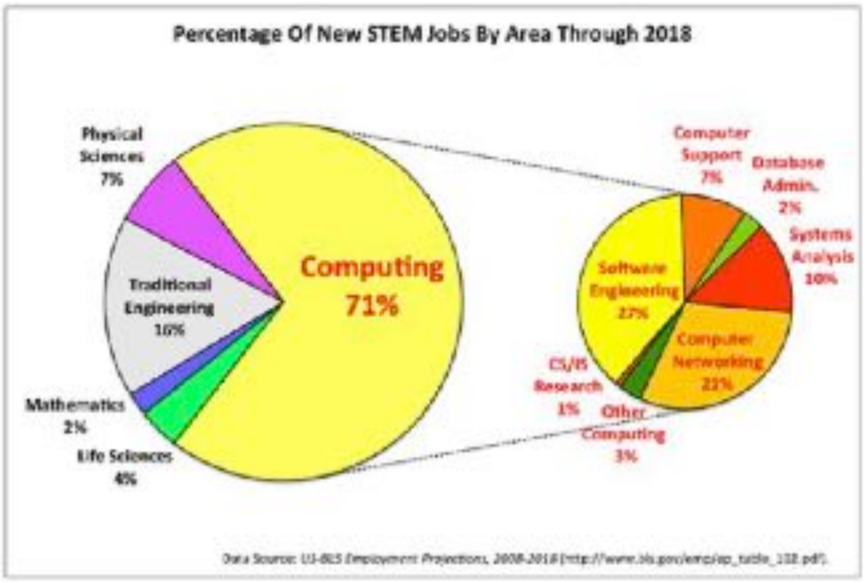

Figure 1: Projection of STEM and Computing job growth through 2018[5]

In the past four semesters and two summers the software engineering module has used a 3-D animation programming environment called Alice. Alice was developed at Carnegie Mellon University under the directorship of the late Dr. Randy Pausch (of the "Last Lecture" fame) as a means for teaching middle school students how to program [3]. Alice provides over 1000 objects that can be added to a virtual world. The objects are supplied with various methods that allow an entire object or parts of an object to perform actions, such as move, turn, play a sound, interact with other objects, etc. The objects also have associated properties that can modify an object's characteristics, such as color, opacity, texture, associated sound files, connectivity to other objects, etc.

There are many advantages associated with using the Alice environment. Alice is free and can easily be downloaded from http://www.alice.org and installed on any Windows or Mac based computer. Alice can even be run directly from a small memory stick in that its memory footprint is only about 360 MB. Alice.org maintains a comprehensive set of class-tested, slide-based tutorials from the "getting started" level to the very advanced level. Alice.org even provides email problem and question support.

The Benefits of Programming in Alice for all Engineering Students

In addition to the importance of having some experience with software development, there are many other important programming constructs that are useful in helping all student engineers appreciate and develop logical thinking skills about complex problem solving and process development. These features include:

- Sequential flow – the concept whereby one step is executed completely followed by the next step, and so on. This concept helps student engineers build an understanding and appreciation of process flow, ordering and the interdependencies of process steps.

- Conditional flow – the concept whereby one of several alternative process paths is executed based on a set of run-time conditions. This concept helps student engineers understand contingencies and the need to plan for all possible process conditions and outcomes.
- Iteration – the concept whereby a set of process steps is repeated over and over again until a specific exit condition is achieved.
- Concurrency – the concept whereby multiple subprocesses are executed at the same time. This may be needed to satisfy requirements of a process, for instance a child proof container may require that one must press and twist the lid concurrently to open. Concurrency may also allow for faster execution of a process.
- Encapsulation – the concept whereby information or details about an object are hidden from other objects and can only be accessed through the object itself. Encapsulation helps promote subprocess abstraction or simplification and assists with the breaking of very large problems and processes into smaller pieces, using a divide and conquer strategy.

In addition to the above process and problem solving skill building, we hope it can help improve recruitment, retention and success rates for introductory computer science and software engineering courses. Although retention and success in introductory computer science courses is not good, as low as 50% at some schools, it is even worse for women[8]. There has also been a decline of 80% in the number of female Computer Science majors between 1998 and 2004[4]. At our university there is no evidence of a reversal of this decline in female enrollment and retention – despite a nearly doubling of total enrollment between 2002 and 2011 it is still rare to have a female student in an upper division software engineering course. Women avoid Computer Science and have lower retention rates than males. Women are the most frequently studied groups at risk in CS1 (CS1 is a generic term used for the first computer science course offered to majors) courses[1, 2, 6, 7]. In these studies, the lower attraction and retention has been attributed to early courses being "overly technical" with little room for creatively[1, 7]. For those student who go on to take a first programming class, Alice experience can be beneficial in increasing the likelihood of success. Without Alice, at risk CS1 students average a C grade and only 47 percent go on to take CS2, with Alice the performance and retention is greatly improved, with CS1 students averaging a B grade and with 88% going on to take CS2[4].

Alice seems to address the issue of programming as being "overly technical". With traditional programming languages students are required to learn complex vocabularies and grammars, in contrast programming in Alice involves dragging and dropping of well formed and valid program statements. Modification of these statements is done by selecting options from dropdown menus. It is impossible to produce the dreaded compile error and very unlikely to produce a program that doesn't run the very first time. This eliminates of the need to memorize a vocabulary and grammar, which subsequently speeds up the opportunity to experience creative expression. It should also be noted that the wide selection of more than a 1000 animation objects with tens of 1000's of methods allow people of practically any interest to find something of interest to animate. For instance there are ballerinas, football players, farm animals, amusement park objects, space objects, plants, cars, trees, flowers, cell phones, etc.

Many students prefer a visual and hands-on learning styles, the author believes that this is even more common among engineering students. Alice addresses this by having a very visual and hands-on development environment. Rather than writing code, compiling it, producing test cases, and so on, the programmer simply selects what they want and clicks the run button to immediately see the result.

Description of Alice

Alice follows a strong object-oriented paradigm similar to popular programming languages like Java, C++, C# and other currently popular languages. If a student is interested in continuing their education in software engineering or computer science, having this object-oriented knowledge and experience is a definite plus. Unlike typical object-oriented programming languages students do not need to learn a programming language vocabulary and grammar. Often with as little as five minutes into a "getting started" tutorial, students are productive and off on their own. Although they don't need to learn the vocabulary, the language is similar enough to a "real" programming language that they kind of get use to it. One of the current enhancements to Alice is to make the language even more similar to Java[4].

Another significant advantage of Alice is that the Integrated Development Environment (IDE) allows for immediate feedback by running the animation, which enables students to try concepts and immediate see how they works. In conjunction with trying concepts, there is an undo button that allows methods and objects to be easily backed out if the desired functionality is not achieved.

Details of the Alice Environment

A Virtual World

The development of an Alice program starts with the programmer selecting a template world. This provides a background to which objects can be added. One of six different templates may be selected by a mouse click. With the template selected, the programmer can start adding objects. See Figure 2 for the template worlds.

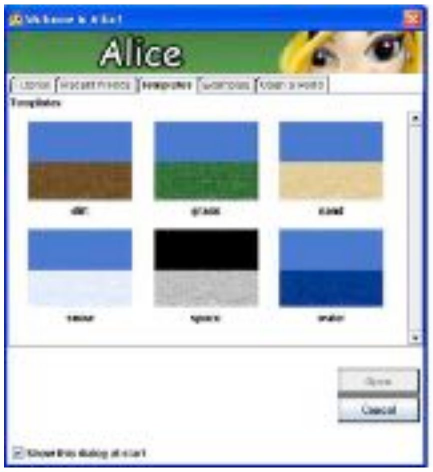

Figure 2: Template worlds.

Object Galleries

Similar to many CAD systems, objects are organized in a hierarchy of galleries. Within a level, galleries are organized alphabetically. The top level gallery contains over 35 subgalleries, such as Amusement Park objects, Animal objects, Beach objects, Old West objects, Skate Park objects, Sport objects, etc., Figure 3. The programmer navigates to a low-level gallery by double clicking on the gallery Figure 4. Objects are added to the world with drag and drop operations. Objects can be positioned and posed by clicking a control and moving the mouse. These position and posing controls allow an object to be place vertically, rotated with respect to the X, Y and Z planes, resized, and replicated Figure 5.

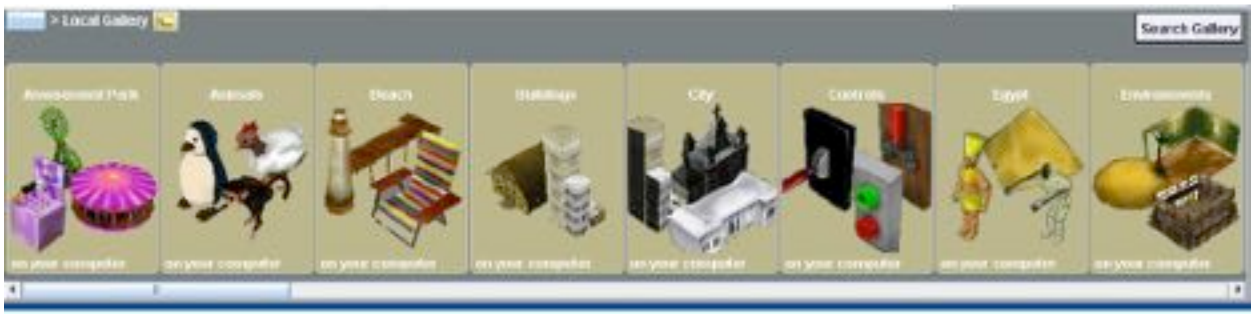

Figure 3: Top-level Object Gallery.

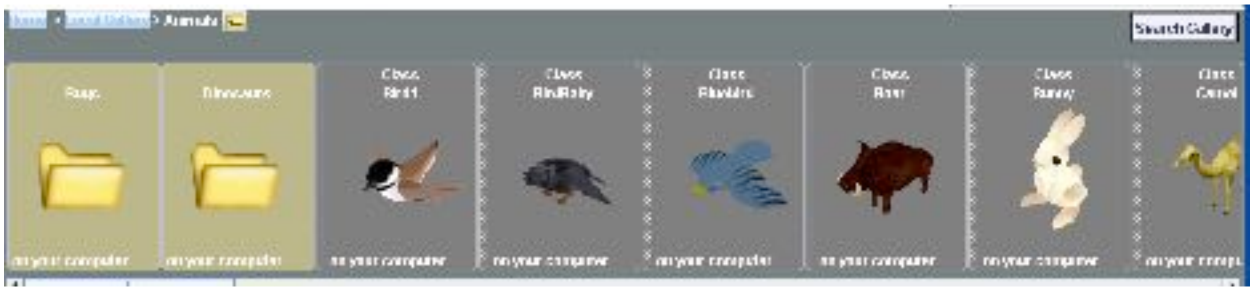

Figure 4: Second-level Gallery – Animals.

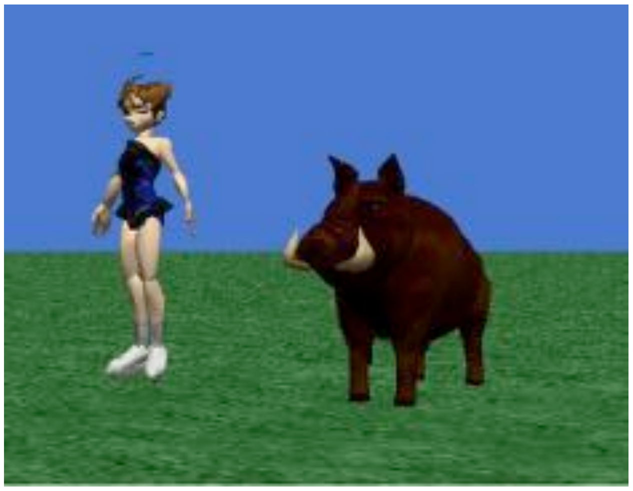
Figure 5: Virtual world with Ice Skater and Boar objects.

There are also three camera controls that allow the programmers to set and save camera positions. The camera can be moved and tilted forward and backwards, left and right, and up and down. Once one or more objects have been added to world the programming of the animation can start.

Programming in Alice

True to the object-oriented nature, most Alice objects are aggregations of other objects. For instance a ice skater object is an aggregation of LeftLeg, RightLeg, UpperBody, and Skirt. The LeftLeg consists of a LowerLeg and a Foot. This is shown in the screenshot in Figure 6.

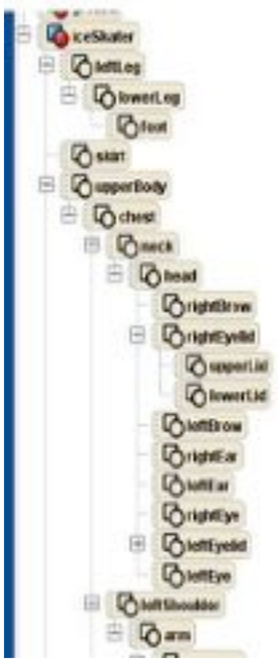

Figure 6: Screenshot of an ice skater object, which shows her as a hierarchical aggregation of other objects.

Each object contains a collection of methods and properties. Methods are action verbs that allow an object to perform an action such as move, roll, spin, play a sound, etc. See figure 7 for some of the methods associates with the ice skater object. Properties are adjectives that allow objects to change color, texture, visibility, etc. These methods and properties can be applied to a top level object, such as the entire ice skater, or one of the lower-level objects such as right eye lid.

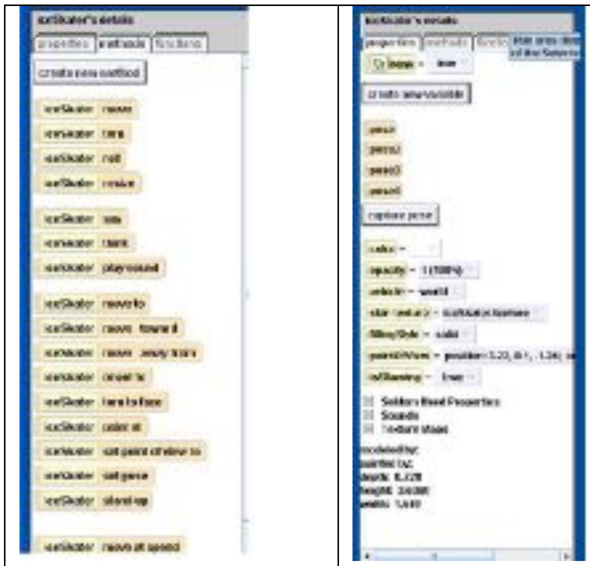

Figure 7: Methods (left) and Properties (right) of the ice skater object.

For a more in depth description of how to program in Alice the reader may want to refer to the fine tutorials at http://www.cs.duke.edu/csed/alice/aliceInSchools/workshop08/tutorials.php.

The Software Engineering Assignment

This section describes the Software Engineering module and its assignment. Students were allowed to form teams of three. The instructor spent about 15 minutes introducing what Alice is and playing a selection of three or four animations from previous semesters. While the animations were running, the instructor pointed out different required features that must be satisfied to earn full credit on the project.

After this introduction to Alice, the class read through and discuses the eight project requirements, while examples from the animations that were just played were reiterated. These requirements include:

1. The World shall contain at least 3 scenes.
2. One of the scenes shall contain at least two characters who engage in a dialog. A sample dialog might be
    - Character 1: What is your favorite engineering discipline?
    - Character 2: (Your answer)
    - Character 1: Why do you like that discipline?
    - Character 2: (Explains why)
    - Character 1: Where did you learn about [name of discipline]?
    - Character 2: (Response)
    - You can use a different dialog instead, but it needs to be engineering-related and involve at least 3 questions with responses.
3. In each scene, the world shall contain at least 1 building, 2 characters, and 3 props (trees, flowers, hockey pucks, skateboards, cell phones, aircraft carriers, etc.).
4. In each scene, two characters shall each move to at least three new places.
5. In each scene, two characters shall move parts. For each character, there must be three body parts moved with each part having at least three movements.
6. At least one scene shall contain an Alice vehicle - something that carries another object. The carried object must enter the vehicle in one scene and leave the vehicle in that or some later scene.
7. In each scene, there shall be at least two camera changes to both new positions and new angles. Each of the angle/position pairs must be distinct from the others.
8. There shall be at least two additional, team-selected, significant features such as events, scene fades, or sounds.

At this point, the teams are coached through bring up the Alice environment and a “getting started” tutorial. This particular computer lab contained dual monitor computers, so the students could follow along the tutorial on one monitor while trying Alice on the other monitor. This first tutorial only takes about 10 minutes and covered requirements 2, 3, 4, and 5, which are the easier ones. The students were given about 15 minutes to work to start populating their worlds with objects and start on the first set of requirements. Following this initial hands-on time, a second demonstration, lasting about 10 minutes, was presented that covered the remaining, more difficult requirements. It should be noted that most groups were seriously engaged in their projects and there were frequent spontaneous outbreaks of laughter, which was judged to be a good sign. The final 30 minutes of the first day’s lab the student were free to work on their projects and ask questions. Each lab section was limited to 10 groups of three, so with a lab assistant there was generally little delay in getting around to answer any questions that came up.

The lab on the second week of the module consisted of a 20 minute presentation (propaganda) of what software engineers do for a living and the career outlook, salary and placement rates. Following this presentation, the students were given free time to continue with their projects. About 15 percent of the groups satisfactorily finish the lab this second day without any out of class time. The other groups required or choose to put in additional out of class time. It is obvious that some groups put in substantial time to go far and beyond just satisfying the requirements. One such example will be demonstrated at the conference.

Conclusions and Observations

Overall the faculty has been very happy with the outcomes of this Software Engineering module and in general the Introduction to Engineering Projects course. Students report that they find the fast paced, hands on introduction covering the seven engineering disciplines of value. We don't anticipate significant changes in the near term.

With respect to the Software Engineering module, we general have one or more General Engineering students per section inquire about changing there major to Software Engineering. This may be due to an epiphany in the discovery of what they really want to major in or it could be due to dissatisfaction in there currently declared major. We would imagine that other disciples also have similar recruitment experiences. We do not have data indicating whether we have seen a net increase in any particular major due to this module.

Currently, the course has a perquisite of an Introduction to Engineering, which targets helping students transition from high school into our college of engineering. The argued benefit of having students take the Introduction to Engineering Projects course their first semester is that it might help more of them make informed decisions regarding their choice of major one semester earlier. At this moment we have a logistical issue that makes this difficult – we don't have enough faculty to teach both Introduction to Engineering and Introduction to Engineering Projects in the same semester as many of the instructors are teaching these courses on an overload basis.

Although the "summer camp" version of this course has only been running two summers, there seems to be a very rapid growth in interest. We have been fortunate to have some funding to offer scholarships to students from underrepresented groups. We feel that this might be a cost effective vehicle to greatly increase the interest and recruitment of underrepresented students into the engineering field. Despite our best efforts, on our campus Software Engineering has one of the lowest levels of underrepresented students. Hopefully, this course will help with this issue.

The two weeks per each of our seven engineering disciplines fits very nicely into a 15 week semester. Our college of engineering has been experiencing steady grow and there are plans to add at least one additional engineering major in the next couple of years. Adding an eighth module will require adjustment of the course schedule.

Alice 3.0 has been in beta testing for many months and includes many new features. Once version 3.0 is released we will need to evaluate the current requirements of the Software Engineering module's to incorporate some of these features. We have chosen to stay with the older 2.2 version to minimize any difficulties the students might experience with a less stable, beta environment.

Bibliography

[1] AAUW. (2000). *Tech-Savvy: Educating Girls in the New Computer Age*. American Association of University Women Education Foundation, New York.
[2] Camp, T. (1997). The incredible shrinking pipeline. Communications of the ACM, 40(10):103-110.
[3] http://www.alice.org retrieved 8/09/2011.
[4] http://www.alice.org/index.php?page=what_is_alice/what_is_alice, retrieved 8/09/2011..
[5] http://cs.calvin.edu/p/ComputingCareersMarket retrieved 8/09/2011.
[6] *inroads* (2002). (The ACM SIGCSE Bulletin). Special issue: Women and computing, Volume 34, No. 2.
[7] Margolis, J. and Fisher, A. (2002). *Unlocking the Clubhouse: Women in Computing*. The MIT Press, Cambridge, MA.
[8] Sloan, R. and Troy, P. (2008) CS 0.5: A Better Approach to Introductory Computer Science Majors. SIGCSE, March, Also available on-line from http://www.cs.uic.edu/~sloan/my-papers/SloanTroy-sigcse08.pdf retrieved 8/09/2011.

Mike Rowe is a Professor of Software Engineering at the University of Wisconsin-Platteville. Prior to teaching, Mike worked in the software industry for 25 years. He remains active in industry through his consulting at Esterline Control Systems – AVISTA. This company does software and system engineering work on mission and life critical avionics and medical systems. He has a B.A. degree from the University of Minnesota Duluth in Psychology, Ph.D. in Physiological Psychology with a minor in Statistics, a Ph.D. in Computer Science, and an M.B.A. In addition to the study of Software Engineering, Mike enjoys high performance computing and statistical modeling.