Course Title: SDE 130 Introduction to Object Oriented Programming

Course Leader: David Maruszewski

Expected Learning Outcomes for Course

- Evaluate and understand the benefits and challenges associated with an object-oriented analysis design approach to software and project development
- Identify the key concepts used in object-oriented development including inheritance, encapsulation, data types, control flow, polymorphism and programming techniques
- Indentify design patterns in terms of participating objects and classes and the roles they take on relative to the problem design solves
- Investigate and evaluate OOA&D (object-orientated analysis and design) tools, methods, and models that are available and currently used in business practice
- Design an algorithmic, object-oriented solution that meets the specification of a programming problem
- Document OOD (object-oriented design) diagrams that meet industry standards

Assessment

(How do students demonstrate achievement of these outcomes?)

Students are required to complete a final project which is applied against a grade sheet. This project was created to test skills gained throughout the course. The sections pertaining to Flowcharts, Storyboards, "two player" and "piece" are evaluated.

Three exams are issued to help confirm the findings of the project grade. Questions pertaining to loops, methods, events, and math application are evaluated.

Validation

(What methods are used to validate your assessment?)

Currently, all grades sheets are held for two semesters and composite data is used to show trends. When this course reaches three sections per semester, a spreadsheet will be created in order to track outcomes. The exams should help verify or contradict findings.

Results

1. Math applications are an issue. High school math is all that is needed for this course, but the application of that math is hard for the students to grasp.
2. Computer concepts are easily grasped when using Alice. Although, students may not be able to articulate them well. They are understood, but not defined by the students.
3. Using programming to control objects in an animated environment is learned well in a simple format.
4. Using many rules in a common animation confuses students and their programming suffers.

Follow-up

(How have you used the data to improve student learning?)

1. I have to re-teach some math in this course. I have spoken with some high school officials at advisory boards in hopes to improve this.

2. The Alice program is good at teaching students how to make programs but not theory. I have to follow up with that. Articulation is needed through tests or through the lecture sections.
3. The Alice program is good at teaching students how to make programs but not theory. I have to follow up with that in class.
4. Breaking things up in "bite-size pieces" helps. I'm trying to work with them on bigger problems. When something doesn't work, I can show them how to get out of it. This is still in trial. Students like things to work perfectly on the first try. It's hard to show them that this usually isn't the way it usually unfolds in reality.

Budget Justification

(What resources are necessary to improve student learning?)

Alice works reasonably well for what we are trying to do, but another software piece might be better in the future. Alice is not being supported well. It may not be able to handle emerging programming issues. As of now, there are some things that it does not teach well.