ASEE AMERICAN SOCIETY FOR ENGINEERING EDUCATION

Paper ID #36332

The First-Year Computer Science Experience Project

Prof. John Cole, The University of Texas at Dallas

I'm an Associate Professor of Instruction at the University of Texas at Dallas. I had taught part-time at Collin County Community College for three years, and prior to that, at Illinois Institute of Technology in the mid-1970s, which is also where I earned my degrees. Before joining the full-time faculty at UTD in Fall 2012, I had taught part-time for 13 semesters. I have been a software developer for many years, working on projects as diverse as a SNOBOL4 compiler, a DATABUS compiler, a word processor, the operating system for an early computer, statistical analysis of insurance claims, telecommunications, and embedded programming. I have used a variety of languages, including Java, C/C++, C#, Visual Basic, Databus, Intel assembly language, and many others. Areas of interest include Human-Computer Interaction and computer science pedagogy.

Full Paper: The First-Year Computer Science Experience Project

Abstract

Many universities require an orientation course to introduce students to the broad discipline of computer science or engineering. At my school this orientation course is offered only in the fall term, and is required of all freshmen declaring computer science or engineering as a major. This paper deals with the project for the Computer Science version of the course.

A team project, in which students write or design a program, or build something related to computation, should be part of any such course. However, incoming freshmen have widely varying degrees of programming background and expertise, from none whatsoever to the equivalent of three semesters of introductory courses. Those with programming experience have typically been exposed to Java, and maybe C++ or Python, so the project cannot be language-specific. Such a project must be difficult enough to give students a sense of accomplishment but not so difficult that they give up. It must also give students a sense of what it is like to do actual computer science and software engineering. It must allow for some level of creativity without being too open-ended. Basic guidelines for such a project are:

- Students with varying levels of experience must be able to specify, design, and possibly implement it in four to six weeks.
- Projects must involve sufficient work that the entire team must participate.
- Projects cannot rely upon extensive programming knowledge, but must involve problem-solving skills.
- Projects are done in phases that build upon each other and are graded separately.
- Each phase stands alone as much as possible so that difficulties in an earlier phase do not insure a bad grade in a later one.
- The size of the project team is four or five students.

Learning objectives in assigning such a project are: Learn about basic software design tools such as flowcharting and pseudocode. Learn how to work with a team. Learn how to come up with an idea and refine it into a software specification.

Instructors for this course have tried various kinds of projects, from programming to cross-discipline projects involving hardware and software, to papers, and have some data on what works well and what does not. My colleagues and I possess written student feedback on these different types of projects from which I have extracted favorable and unfavorable comments. In this paper I draw from my own experience of having taught multiple sections of this course since 2013 as well as feedback from other instructors to determine the kinds of projects that best meet our objectives.

Introduction

Almost all software written today requires a team of people. Therefore, it is vitally important for students to learn the skills involved in working with others to achieve a goal. Since introductory programming courses focus on individual work and only later courses will require students to

work in teams, our CS1200 course, described below, needs to have a good introduction to these skills. This paper explores what kinds of projects lead to the best outcome with respect to the following criteria:

- Did all students participate in the project more or less equally? Could the project be divided into enough roughly equal parts that each member of the team did the same amount of work?
- Could the project be accomplished with minimal to no programming knowledge or skill?
- Did the project require problem-solving, both at the technical and organizational levels?
- Did the project encourage creativity rather than just following a path already laid out?

Student Population

Every incoming freshman who declares computer science as a major must take this course. It is offered only in the Fall term. We can be reasonably sure that students are proficient in math and that they got good high school grades and high SAT scores. They were from the usual demographic to be found in a school like ours: ages ranged from 17 to 19 and they were from many racial and cultural backgrounds.

Course Description

Here is the catalog description: "**CS 1200 - Introduction to Computer Science and Software Engineering** (2 semester credit hours) Introduction to the computing professions; overview of Computer Science (CS) and Software Engineering (SE) curricula, connections with Computer Engineering, other Engineering and Computer Science fields, and Arts and Technology programs; problem solving and other skills needed to succeed as a CS or SE major. Introduction to quantitative methods; team projects designed to replicate decision processes and problem solving in real-world situations; additional preparatory topics for CS and SE majors."

The key course learning objectives are as follows:

- Awareness of the areas within CS & SE and curricula
- Understanding of basic logical thinking and problem solving
- Capability of high-level solution design for simple algorithms
- Ability to work with teams

Team Formation

We tried two ways of creating teams.

One was to assign students to teams based upon who they sit with and instructor observation of classroom interactions. This had the advantages of being able to balance male-female ratios in the teams, and having students get to know other students they might not have met otherwise. This is also closer to the way the work world operates, where employees work with the other people the company has hired.

The other was to let the students self-select based upon whom they know, their roommates, or simply sign up for a random group if they don't know anyone. The instructor reserved the right

to reassign people if groups were too small, someone dropped the course or simply didn't show up, or by student request. This appeared to form more cohesive groups, better able to work together.

Projects we Tried

When we created the ECS1200 course, the idea was to have students from all of the engineering disciplines, including computer science, in the same class. Since teams could have students from any discipline, and usually did, they were allowed to choose from the following projects:

- A bridge made of spaghetti and hot glue
- A motion-activated nerf gun using an Arduino for sensing and control
- A mechanical arm that would move objects from one bucket to another
- A computer game using Alice [3]

After we changed the course from cross-disciplinary to computer science only in the Fall 2015 term, we tried a variety of new projects from only our discipline. These included:

Come up with your own Raspberry Pi project.

Design a computation device that doesn't rely upon electronics. It must be able to add two two-bit numbers and produce a result.

Write a paper on number systems, including at least three number systems other than the Arabic numerals we all use. Explain how arithmetic operations are performed in each of these systems and give examples. This was an individual project, so the course requirement of a team project was met another way.

"Design your Process for Becoming a World-Class Computer Science Student." This was based on work by Steffen Peuker and Raymond Landis [1]. This was also an individual project.

Choose from a list of projects supplied by the instructor.

Methodology

Since there was a wide range of coding knowledge and ability, I ruled out programming projects of any kind. I also found that asking the students to come up with their own project had two problems. Either the project was so simple it could be completed in a day, or it was so complex it would have required a large team and a year or more to finish.

The non-electronic computation device was inspired by a paper by Paul Fishwick [2] and was fun, but students mostly looked things up on the Web rather than doing creative work. I also saw that these did not lend themselves well to team effort.

Papers were good, but individual papers didn't fit the course objective of a team project. I also found that some students could not be convinced to take the paper seriously, and for research papers plagiarism was a problem. Writing skill was another issue. Objective grading was difficult, since students had different ideas as to what was required and different levels of detail in what they wrote.

Finally, we settled on letting students select a project defined by the instructor. This project was done in four phases, each independently graded. The result would be a complete project plan including the first three phases but not the presentation. The phases were:

1. Phase 1 was to form a team and choose a project from the list provided. Turn in a document containing the following:
    a. Group Name
    b. Group number
    c. List of all group members including e-mail addresses
    d. Name of group leader
    e. Project Idea (Bullet point 1 from the "Design an App" slide)
    f. A one-paragraph description of the major functions of the app.
2. Phase 2 built on phase 1 by adding the following additional information:
    a. A few paragraphs describing what the app does and how it would be used for some common cases.
    b. A listing of all the requirements of the app. These are relatively large-scale functions. "Creating the user interface" is not a task, since various functions will have their own interfaces. Spend some time on this; it is not as easy as it seems.
    c. Potential problems meeting the requirements.
    d. Break the development of the app into subsystems.
    e. Assign each member of the group one or more subsystems. Write a short paragraph on what each subsystem does. There must be at least as many major subsystems of the program as there are team members. If there aren't, you haven't thought through the problem well enough. Note: a login screen that takes some sort of ID and a password is a very small task, and will not be allowed as a "major subsystem." Likewise, a "back end" will not be allowed as a subtask.
3. Phase 3 included everything from the previous phase along with the following:
    a. The sub-problem you selected to discuss, from Phase 2. This will be different for each group member, and will be graded individually. Go into enough detail so that I will understand your pseudocode and diagrams, below. This will typically require a paragraph or more.
    b. The platform(s) on which your part runs.
    c. Finally, you must complete the following for **your sub-problem only**: (Your document must have these parts **in this order**.) Pseudocode and flowcharts that are entirely linear, without branching or looping, are probably too simplistic to meet requirements ii and iii.
        i. A user interface prototype
        ii. Pseudocode for the sub-problem, or a significant part of it
        iii. Flowchart or UML Activity Diagram showing flow of your sub-problem at a high level.
        iv. The kinds of data your sub-problem needs to store to accomplish its task.
4. Phase 4 was a presentation to the entire class, with an introductory slide, one slide for each student's subsystem, and a closing slide. Students were allowed one minute for the team leader to introduce the team and describe the overall project. Each team member

then presented his or her subsystem. Finally, the team leader would describe the conclusions. The final slide was a group photo.

The project ideas supplied by the instructor were:

1. A game to explain a class topic.
2. A student response system similar to the Clicker app.
3. A CS1200 class communication app.
4. Assignment and study tracker.

The instructors found that there was a reasonable distribution over the four projects, although the game and the assignment and study tracker tended to be slightly favored.

Results and Discussion

The quantitative data in Table 1 are limited because, as you can see, not every student completed the course evaluation, and not every student who completed it wrote comments. Since there was no question specifically about the project, I had to rely on the written comments. Not all written comments mention the project, so of those that did, I looked for favorable or unfavorable comments.

Table 1. Favorable and unfavorable comments on various projects.

Class/Year and Project Type	Total Students	Total Evals	Favorable	Unfavorable
2015 -- Pi/Arduino	126	30	8	18
2016 -- Arduino or non-electronic	128	30	6	5
2017 -- Design Your Process	130	31	3	5
2018 -- Design an app	168	29	3	4
2019 -- Design an app	182	99	13	4
2020 -- Design an app	187	31	8	3
2021 -- Design an app	108	54	5	1

Favorable comments were generally along the lines of the following: “The assignments and the project should remain the same. I enjoyed doing them.” “The final project was appropriate.” “I think the project was a great thing to have in the course.” “I enjoyed the final project and the way it forced me to go out and acquire new skills in a team.”

Then there were unfavorable comments. “The way the project deliverables are graded needed more explaining in my opinion.” “End group project was too oriented towards people with programming experience rather than people who did not.” “Overall, the [project] tasks given, especially with a group with members from different levels of CS, felt too advanced and I was not able to learn much.”

In 2015, the Pi/Arduino project was viewed very unfavorably. Most comments concerned not having enough programming background to do it effectively, issues with hardware, and not being able to participate because someone else on the team with more knowledge just took over.

The following year was better when we included non-electronic computation devices, since many students took this option. However, the issue of finding too much online became more serious.

The "Design your Process" paper in 2017 was definitely not a favorite, although few students chose to comment on it. One comment I saw occasionally during the grading process was that some students had a hard time taking it seriously. A few commented to me during office hours that it was helpful but unclear as to what to write.

The next four years were various iterations of the "Design an App" project. Things that varied over the iterations were, among others, the projects, as noted above; penalties for late work and nonparticipation; and the level of detail for describing the projects. Earlier iterations included less detail, although it was better to leave more design decisions to the students if possible.

Conclusions

The project is an important part of the freshman seminar and must be chosen carefully. Those that require no actual programming are best. Projects that are done in phases rather than all or nothing are also better received. Projects that involve too much problem-solving at a low level were also rated more poorly by the students. For example, in Arduino and Raspberry Pi projects, students found debugging a circuit frustrating.

Acknowledgements

Special thanks to Prof. Janell Straach for getting me involved with teaching CS1200 in the first place. Thanks to Prof. Timothy McMahan for invaluable discussion of the project and his ideas on how it should be structured.

References

[1] Workshop: "Design Your Process of Becoming a World-Class Engineering Student"– A Powerful Project for Enhancing Student Success by Steffen Peuker and Raymond B. Landis. Presented at the 6th First Year Engineering Experience (FYEE) Conference, August 7 – 8, 2014, College Station, TX

[2] Finding computation in Everyday Objects. https://en.ava360.com/finding-computing-in-everyday-objects-dr-paul-fishwick-at-tedxutd_ad4ed1304.html

[3] https://www.alice.org/