Using Course-Long Programming Projects in CS2

Joseph A. Turner
University of Utah
Department of Computer Science
jaturner@cs.utah.edu

Joseph L. Zachary
University of Utah
Department of Computer Science
zachary@cs.utah.edu

1. ABSTRACT

A typical CS2 course has two goals that often work at cross-purposes. One goal is to teach students how to apply a variety of software engineering skills to create solutions to real-world problems. A second goal is to teach students the theory and practice behind classical algorithms and data structures. The use of small, short-term programming assignments, however, tends to sacrifice the first goal in favor of the second. We successfully experimented with solving this problem by organizing a CS2 course around a programming project that spanned an entire term. This paper describes the project, our experiences in using it, and the reactions of the students.

1.1 Keywords

Education, CS2, programming assignments, software engineering, code maintenance.

2. INTRODUCTION

A typical CS2 course has two goals that often end up opposing one another. One goal is to teach students who have already learned basic coding skills how to design, analyze, implement, test, and debug programs on a larger scale. A second goal is to teach students the theory and practice behind classical algorithms (such as sorting and searching) and classical data structures (such as linked lists, binary trees, hash tables, and graphs).

The way in which CS2 programming assignments are commonly structured, unfortunately, tends to elevate the second goal at the expense of the first. Instructors typically determine whether their students have mastered the concept behind an algorithm or data structure by assigning a program that exercises that concept. To facilitate grading and to permit the coverage of a variety of topics, these assignments are generally small programs that are targeted to illustrate a single concept in isolation from all the others. They are completed over a period of one or two weeks, and are then discarded by their authors.

SIGCSE '99 3/99 New Orleans, LA, USA

This way of structuring programming assignments has two major drawbacks. Because they are small, these programs appear as toys to the students and do not have a strong connection to the world of software applications with which students are already familiar. As a result, some students have long-term trouble applying even well understood computer science concepts to real-world problems.

Because the programs are completed quickly and then discarded, students are not required to maintain, review, or even fully test and debug their programs. As a result, students never develop good debugging skills, are never forced to live with the consequences of a poor design, never fully appreciate the tradeoffs inherent in different ways of implementing abstract data types, never experience the power of code reuse, and never even learn the value of commenting source code.

We experimented with solving these problems by organizing a CS2 course around a programming project that spanned an entire term. The students implemented a basic day planner application with which a user can manage a schedule of events and display them on a web page. We decomposed the project into five steps, each focused on a traditional CS2 topic. However, we made sure that each subsequent assignment depended on the previous one.

We found that using a course-long project served to overcome the disadvantages discussed above. Additionally, students were enthusiastic about working on a program that they would be able to use when the course was over. Since the course has ended, we have been asked by a number of students how to add functionality to the day planner, and we have received descriptions of features that other students have added on their own.

The remainder of this paper is organized as follows. Section 3 describes the course in which we employed our experiment. Section 4 describes the Day Planner project in detail and highlights ways in which the project enhanced the learning experience. Section 5 discusses ways in which the project could be extended, section 6 discusses related work, and we conclude in section 7.

3. THE CS2 COURSE

The CS2 course that we used for this experiment is the second in a two-quarter introductory sequence. The preceding CS1 course covers such C++ constructs as variables, conditionals, loops, procedures, console I/O, file I/O, and structures. We began our CS2 course by discussing C++ classes and then introduced pointers, linked lists, sorting, binary search trees, hash tables, graphs, inheritance and polymorphism. We also devoted some time to explaining the personal software process to help enhance students' software engineering skills [4].

The CS2 course described in this paper was taught during the summer quarter of 1998, and was an accelerated version of the regular CS2 course. As a result, it had fewer assignments than a typical quarter-length CS2 course. In Section 5 we discuss ways to lengthen the assignment to fill a full quarter or semester.

Sixty-seven students completed the course. Students attended two 90-minute lectures and one two-hour lab during each of the nine weeks of the summer quarter.

The students taking the CS1/CS2 sequence are expected to have some prior programming experience, although not necessarily with C or C++. CS1 and CS2 are required by several departments in the College of Engineering, so the sequence is not filled exclusively with computer science undergraduates.

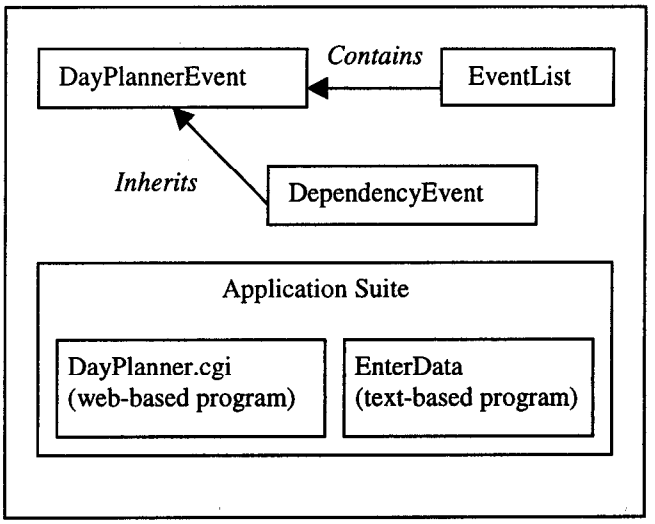
4. THE COURSE-LONG PROJECT

The course-long project described in this paper is a day planner with which users can maintain a list of events. For example, a user can insert into the list of events a scheduled meeting or an upcoming test. The day planner also provides a mechanism for creating dependencies among events. (A dependency might assert, for example, that one needs to study before taking a test.) Besides allowing events to be entering and maintained, the day planner also provides some manipulation features. For example, the user can sort events by date or by description. Some of the students also added a feature that queries the operating system for the current date and then searches for events that occur during the current week. These events can then be emailed to the user or to a group of users.

There were three major components to the project: the events themselves, a list of events and a suite of applications. The students created the events and the list of events. We gave the students the application suite, but they were required to modify the applications it contained.

Events were broken down into two subcategories, regular events (DayPlannerEvent) and events that depend on other events (DependencyEvent). There were also two different interfaces, one text-based and the other web-based. Figure 1 gives a graphical description of the components and their relationships to each other.

Because we were able to decompose the complete project into five consecutive assignments, we were able to maintain



the focus of traditional programming assignments while adding the benefits of a course-long assignment [3]. Students were not overwhelmed by having to implement the whole project at once. They were able to concentrate on each individual component until it was completed, then move on to the next component.

4.1 The application suite

As the World Wide Web (WWW) becomes more popular, students are becoming more interested in learning how to develop web-based, as opposed to text-based, applications [6]. One of the problems with traditional text-based projects is that they do not look like the computer programs with which students are familiar. In general, students are less interested in writing programs that are exclusively text-based than those that are graphically-based [7,8]. Text-based programs do not have the look and feel of a modern program with windows, text entry areas, buttons, slider bars, etc. Results from more traditional programs are typically displayed with simple text and not with tables or graphs or other modern methods. Thus, we decided that using HTML and the WWW would help to spark interest in the programming projects, and would give them the look and feel of modern programs.

We provided our students with two separate interfaces to the application suite. We wanted to have a web-based interface to create interest in the programming project, but programming exclusively for the web poses some difficult debugging issues. In order to make debugging simpler for students, we also implemented and released to the students a text-based interface. The two interfaces also served to illustrate one aspect of code reuse; namely, how one set of abstract data types can be used by two different programs.

The text-based application provided simple data manipulation functions. The web-based application was used mainly to display the list of events in interesting ways. However, the user of the web-based application could add and delete events from the list.

4.2 Events

The basic component of the project is an event. For our purposes an event is a C++ class containing:

1. a description: character string
2. a date: a date class we provided
3. a time: a time class we provided
4. a priority: a simple enumeration
5. a link field: character string

Students were given the header file to the DayPlannerEvent and were asked to implement each of the functions and to declare all necessary data members. Because pointers had not yet been introduced to the students, they were required to use static arrays. However, in order to help motivate dynamic memory we required the students to make sure they did not overrun the size of their character arrays. The students also implemented a display function, which took an output stream. With the display function structured in this way, we could capture the string the student wished to display for testing purposes. At this stage of the assignment the day planner supported adding and deleting events. An array of events was used to store events in the text-based application.

4.2.1 Concepts covered

For the event part of the assignment, we introduced students to both using and developing C++ classes. Because we required students to include a date and a time class in their version of the DayPlannerEvent class, they were required to create and use instances of the Date and Time classes. Additionally, students were required to develop the DayPlannerEvent class so they acquired experience in designing and implementing classes.

We found that the project we assigned had three beneficial side effects. The first was that we were able to motivate dynamic memory by requiring students to truncate strings that were too long for them to store. In lecture one student asked, "under what situations do you use dynamic memory?" We were then able to discuss the DayPlannerEvent class and describe how it would be more efficiently implemented if it used dynamic memory instead of static memory. This discussion led to redesigning the DayPlannerEvent class.

The display function was used to show how operator overloading worked. Rather than making a function call to the display function, students could just use the "<<" operator to write to the console or to a file.

Because we required students to update their code a month after they had written it, students had to rely on their comments to understand what they had done. Students that did not comment their code suddenly wished they had, and students that did comment their code wished their comments contained more information that was relevant. To truly understand why code should be commented, students need to debug or modify code that isn't commented well.

We also found that because students had to use the classes they wrote, students learned how to comment what a function does for future uses. For example, one student was having trouble figuring out how to implement a part of their solution that required using the DayPlannerEvent class. We explained to the student that he should search through the comments in his header file to try to find a function that did what he was looking for. After a couple of minutes of searching he proclaimed that he now understood why functions should be commented as to their behavior.

4.3 The list of events

The EventList component of the day planner was a doubly linked list of pointers to events. The EventList was index based: all accesses to the EventList (inserting, deleting, retrieving, etc.) were done relative to a position. Any style of linked list could have been used here. We chose a doubly linked list because the course textbook contained most of the implementation for a singly linked list.

Again we wrote and gave the interface for the linked list class. There were six functions required by the linked list class: constructor, destructor, length, insert, delete, and retrieve. The students were required to implement each of these functions as well as to define their own structure for the nodes of the linked list.

We released a new version of the application suite that used the list to store events. Additionally, a feature that allowed the user to save their list of events was added to the text-based application. The web-based application then read the saved list and displayed the list in a tabular format.

4.3.1 Concepts Covered

In order to implement a linked list the student must understand dynamic memory and pointers, so we started the discussion there. We aimed to make sure that students had perfectly working code so we tested for memory leaks. This required students to have a deep understanding of how to properly use new and delete. A complete lecture was spent describing gdb (the gnu debugger) to help them locate memory leaks and other pointer related bugs.

4.4 Sorting

We required the students to implement three sorting algorithms for this part of the assignment. We used bubble sort, insertion sort, and selection sort. Because we were sorting with a linked list as opposed to an array we decided not to use the more traditional Quicksort or merge sort. For this part of the assignment students were required to add sorting functionality to one of the applications in the suite. Students were also required to modify the header file for their version of EventList.

Students were required to be able to sort by priority, by description and by date. At this point in the project we provided less than and equality operators in the Date and

Time classes to facilitate sorting. It was suggested that students start by writing a swap function for their linked list and then write the sorting functions using the swap function. After adding sorting to their list class students added a sorting option to the text-based application.

4.4.1 Concepts covered

Simple sorting concepts were covered, such as writing a swap function for a linked list. Students were given the choice of implementing bubble, insertion or selection sort. We explored two different designs for the solution. The first was to have the students implement one sorting function and change which comparison they did (compare on time, compare on date, or compare on description). Other students implemented three separate sorts and had to maintain each of the sorts. Thus we were able to show how students should try and reuse code so that they did not have to maintain as much code.

4.5 Dependency Event

The last constrained part of the assignment was to add dependencies to the day planner. Dependency is used here to describe the situation where one event must be completed before a second event can be completed. For example, before one can attend a birthday party one must find transportation to the party. This led to a graph of dependencies. The addition of priorities to dependencies produces a weighted graph.

The DependencyEvent class inherited from the DayPlannerEvent class. We spent an extensive amount of time showing that when we subclassed from the DayPlannerEvent class that the EventList class would accept both DayPlannerEvents and the new DependencyEvent class.

No code was handed out for this part of the assignment. Students were required to modify their existing DayPlannerEvent class to be "inheritance friendly." We expected them to add a virtual destructor, and change the display function into a virtual function.

Students were also required to create a suite of functions in the text-based program for modifying dependencies (for example, adding dependencies to a DayPlannerEvent, thus changing the priority for that event). For the purposes of the assignment students implemented a dependency as a pointer to a DayPlannerEvent and a corresponding priority, which was defined as an enumeration.

The biggest challenge of this assignment was writing the expanded version of the display function. First students needed to have a deep understanding of the object-oriented relationships has-a and is-a because a DependencyEvent had both a has-a relationship and an is-a relationship. The is-a relationship arose because the dependency event was subclassed from the DayPlannerEvent class. The has-a relationship arose because the dependency event had pointers to other DayPlannerEvents.

In order to aid debugging and testing we had students add debugging statements to each of their functions in both the DayPlannerEvent class and the DependencyEvent class. This allowed us to make sure that they had appropriately added virtual keywords where necessary. It also helped students understand what was expected as we released a "trace" of our working version that listed which functions were called and when. For example, when destroying a DependencyEvent one should see a call to both the DayPlannerEvent destructor and the DependencyEvent desctructor.

4.5.1 Concepts covered

Students are required to have a good understanding of a variety of object oriented concepts in order to complete this part of the project. Virtual functions were explored with the display and destructor functions. Calling an inherited function was described by the display function, since to display a DependencyEvent one needs to display the DayPlannerEvent first and then the parts specific to the DependencyEvent. This part of the assignment also prepared the layout of a graph so later parts of the project could explore graph algorithms.

4.6 Free for all

The last part of the assignment was a free-for-all. Students were required to add interesting functionality to the suite of programs. In order to receive credit for this part of the assignment students were required to demonstrate the program to either the professor or a teaching assistant. Here is a short list of some of the more interesting features added.

- Rather than printing out a complete list of the events students created a calendar display. For example, they would display all events for April 1998. Some students expanded the display such that all days of the month were laid out in a traditional calendar format with events marked on the appropriate day.
- We provided a simple function that opened a pipe to send mail. Some students used this function to mail next week's events. The challenging part of this assignment was to figure out how to get the current time and convert it into the provided date and time classes. This part of the project required searching through on-line manual pages to find appropriate functions.
- The original plan for this part of the assignment was to modify the display function for the DependencyEvent class such that it would detect cycles. Thus if event A depends on event B which depends on event A would be flagged as a cyclic dependency. Cycles of multiple lengths were also reported.

5. FUTURE WORK

This section contains a set of possible additions and changes to the assignment.

1. After pointers have been covered, re-implement the DayPlannerEvent to use dynamic character arrays.
2. Provide some more "smarts" to the suite of applications, such as making sure that a graph of DependencyEvents makes sense. For example, if event A has a due date of Tuesday, and event A depends on an event B that is not due until Friday, this would not make sense.
3. Add a queuing mechanism for events. That is, provide a way to list events in the order in which they should be completed.

6. RELATED WORK

[2] discusses the use of a course-long project in a CS1 course. However, our study concentrates on the benefits of a successful course-long project on real world problem solving. For example, one of these benefits is code maintenance. Additionally, our project explores integrating object oriented programming with web-based interfaces, which are not typically covered in CS1 courses.

7. CONCLUSIONS

A course-long programming project has many advantages over shorter independent programming assignments. First, course-long programming projects require students to maintain their code. Code maintenance helps students in four ways: increasing their debugging skills, exploring weaknesses in certain designs, learning the tradeoffs of certain decisions for interfaces, and experiencing the pain of debugging comment-free code. For an accelerated CS2 course we assigned a course long programming that helped develop the students' code maintenance skills [5].

It is not enough for students to demonstrate competency in a computer science concepts; they should also be able to apply the concepts to real-world problems. Most traditional programming assignments abstract the concept being covered to the point that students may completely understand a topic but not are able to apply it to a problem. For example a student could implement a linked list perfectly but not understand how or when to use one.

Because we handed out header files, we were able to automate the testing procedure by writing simple programs to drive the students' classes. Besides being a faster way of testing, automated testing has the important side effect of freeing up the teaching assistants to help students on an individual basis. Having an abundance of individual help available to students was extremely important. Since students were required to correctly complete previously assigned components of the project, many students needed individual help to understand what mistakes they were making.

It is a given that some students will fall behind in their course work. Uncontrollable external forces in students' lives prevent them from getting parts of an assignment completed. In order to cater to students that fell behind we agreed to hand out our part of the solution if they were willing to sign a non-disclosure agreement. Additionally, we were planning to require students to type in the code to force them to read it rather than just copying and using it. However, because the teaching assistants were free to spend time working one-on-one with students, those that fell behind were able to catch up to the rest of the class. We never were asked to give up our solution.

8. ACKNOWLEDGMENTS

Great credit goes to the teaching assistants: John Blanchard (testing), Ron Lenk (memory leak testing, web interface), and David James (one-on-one teaching). Mike Sanders set up the Apache web server that the students used.

9. REFERENCES

[1] Online course material for the web based day planner. http://www.cs.utah.edu/~jaturner/DayPlanner

[2] Bareiss, Catherine C. A Semester Project for CS1 in Proceedings of SIGCSE '96, ACM Press, 310-314.

[3] Holmes, Geoffrey and Smith Tony C., Adding Some Spice to CS1 Curricula Proceedings of SIGCSE '97, ACM Press, 204-208

[4] Humphrey, Watts S., A Discipline for Software Engineering, Addison-Wesley, 1995

[5] Oliver, S. Ron and John Bardbey, A Software Development Process Laboratory for CS1 and CS2 in Papers of the 25th ACM SIGCSE Technical Symposium on Computer Science Education '94, ACM Press, 169 – 173.

[6] Roberge', James, Creating Programming Projects with Visual Impact in Papers of the 23rd ACM SIGCSE Technical Symposium on Computer Science Education '92, ACM Press, 230-234.

[7] Roberts, Eric S. Using C in CS1, Evaluating the Stanford Experience in Papers of the 24th ACM SIGCSE Technical Symposium on Computer Science Education '93, ACM Press, 117-121.

[8] Roberts, Eric S. A C Based Graphics Library for CS1 in the Proceedings of the 26th SIGCSE Technical Symposium on Computer Science Education '94, ACM Press, 163-167.