MODULAR TERM-LONG CS2 PROJECTS

James K. Huggins[1]

Abstract *— We present a model for a term-long CS2 project which emphasizes modular code development and code-reuse while providing the opportunity to cover many traditional CS2 concepts. The project focuses on the construction of a simplified database system. The project is composed of a series of smaller assignments; each assignment requires students to replace or enhance work completed in previous assignments, thereby allowing students to experience the benefits of good modular design. Each project is of manageable size and complexity for both student and instructor; the final product is a program of non-trivial size and substantial functionality. The project can be easily varied from term to term in order to combat plagiarism, yet remains similar enough to administrate reliably We have successfully used this model over the last three years.*

Index Terms — CS2, modular programs, re-use, term-long projects

INTRODUCTION

At Kettering University (formerly known as GMI Engineering & Management Institute), we use Java as our language of instruction for CS1 and CS2, using Sun Microsystems' Java Development Kit (JDK) [5]. Teaching CS2 at our institution presents several unique challenges.

All students at Kettering University alternate between twelve weeks of classroom instruction and twelve weeks of corporate work experience during their academic careers. Thus, every CS2 student experiences a three-month gap between the end of CS1 and the beginning of CS2. During that three-month gap, these students may not use Java at all, or they may have used other programming languages. Consequently, students need an opportunity (albeit brief) to re-acquaint themselves with Java (and our programming environment) before effective instruction can begin.

Students at Kettering University take an average of five courses per term, beginning in the first year. Thus, most CS2 students are terribly busy with projects and quickly become deadline-driven in their approach to coursework. This makes large, term-long projects difficult to execute successfully without creating many small checkpoints that serve as miniature deadlines.

Teaching CS1 in Java in eleven weeks of classroom instruction (reserving the twelfth week for final exams) is a significant challenge. We must cover object-oriented programming in CS1, as Java's object model is central to the language. Adding this material to our CS1 course necessitates deferring virtually all non-essential topics to CS2. In particular, the use of Java's Abstract Windowing Toolkit (AWT) [6] for building graphical applications and applets cannot be covered in CS1, much to the disappointment of our students (who are excited about the prospect of building their own Java graphical systems).

Amidst all of these challenges, we still attempt to cover the usual CS2 topics, including recursion, dynamic data structures, classical abstract data types, algorithm analysis, and further development of the object-oriented programming philosophy. Traditionally this course also incorporates a large-scale project to begin to give students experience in working with programs of non-trivial size.

We describe a model for a term-long project which emphasizes modular program development, utilizes a variety of dynamic data structures, ends in a GUI-driven application, and is still feasible to complete given the above constraints. This model has been successfully used over several successive terms.

OVERVIEW OF THE PROJECT

The term-long project centers on the construction of a simplified database system. Examples of projects used in the past include:

- A dictionary for a simple block-replacement code mapping cleartext words to their encoded counterparts and vice versa.
- A circulation system for a video store with a list of videos which can be checked out and later returned
- A system for maintaining a student's academic history, with lists of courses taken and the corresponding grades.

The term-long project is broken into four smaller projects. Students are given two weeks to complete each

[1] [1]James K. Huggins, Kettering University, Computer Science Program, 1700 W. Third Avenue, Flint, MI 48504-4898, jhuggins@kettering.edu

project. (The remaining three weeks are used for an additional stand-alone program focused on recursive backtracking and a pause for the midterm examination.)

Program 1: Setting The Framework

In Program 1, students develop a simple version of the database program. The program reads the database from a text file (stored in a simple format) and allows the user to search the database for specific records and print the entire database.

Students are required to implement this program using several classes:

- A class representing the database records being manipulated by the system (e.g. dictionary entries, videos, and courses), with associated methods for creating and modifying individual records.
- A class representing a collection of database records, with associated methods for creating and manipulating this collection. For Program 1, students implement this collection as an (unordered) array of records.
- A class representing the driver for the program, which calls various methods from the collection class upon request from the user. For Program 1, all user requests and responses are performed through standard input and output using simple text-based I/O routines.
- Any additional classes deemed helpful by the student.

Students are instructed to design their programs in as modular a manner as possible. In particular, students are warned that they will be revising and replacing these classes as the term progresses.

This program is comparable in difficulty to programs which students are writing at the end of our CS1 course. Consequently, this program can be (and is) assigned on the first day of CS2.

Program 2: Dynamic Collections

In Program 2, students are required to replace the array-based collection class used in Program 1 with another class using dynamically allocated linked lists. Several variations on the linked list theme have been used in different semesters:

- A classical, singly linked list
- A doubly-sorted linked list (where each node in the list has two links, indicating the successor with respect to each sorting scheme)
- A collection of a fixed number of linked lists (e.g. lists of checked-in and checked-out videos)
- A linked list of linked lists (e.g. for the academic history database, a linked list of terms, each containing a linked list of courses taken during that term)

Students are required to maintain the linked lists in an appropriate sorted order. Additionally, students are required to implement new user commands that call for inserting and deleting records into those linked lists. Often, these user commands require insertion and deletion only implicitly (e.g. checking out a video requires a deletion from the checked-in list and an insertion into the checked-out list).

Program 3: More Dynamic Collections

In Program 3, students are required to replace (again) the collection class with another class using dynamically allocated binary search trees. Several variations on the tree theme have been used in different semesters:

- A classical binary search tree
- A doubly-sorted binary search tree (where each node in the tree has four links, two for each tree)
- A collection of a fixed number of binary search trees
- A combination of linked lists and trees (e.g. a linked list of tree structures)

The same ordering scheme used in the previous programs is used to order the binary search tree(s). Students are additionally required to implement new user commands, notably explicit save-to-file and load-from-file commands.

Program 4: Graphical Interface

In Program 4, students are required to replace the text-oriented user interface used in Program 3 with a graphical interface, using Java's Abstract Windowing Toolkit (AWT). Students are not required to add any new functionality to the program; rather, all functionality present in previous programs (which usually involves 8-10 different user commands by this point) must be supported in a graphical framework.

DISCUSSION

The use of term-long projects in CS2 is certainly not original to this paper; many others have successfully used term projects with great success [2,4,9]. This particular framework for a CS2 term-long project has been successfully used for the last three years. Some of the benefits of this scheme are outlined below.

Modularity and Reuse

One of the well-known difficulties of teaching programming is the chasm between traditional, short, self-contained programming projects often used in CS1/CS2 and the large, integrated, continuing projects with which programmers must contend in real-life. While senior-level design courses (*e.g.*, software engineering) can provide students with experiences in "programming in the large," opportunities to gain experience in writing and re-writing programs early in the curriculum is highly desirable, especially for students who are already putting their skills to use in a co-operative work setting.

The model presented above provides multiple opportunities for students to practice modular design. Students are forced to alter one or more classes on three separate occasions. This gives them the opportunity to experience the benefits (or consequences) of their own design choices, as the collection class changes from arrays to linked lists to binary search trees, and as their interface class changes from text-based to graphical operation. Anecdotal evidence suggests that students do seem to get the point (one way or the other).

Additionally, the continual reuse of existing code from program to program gives students the opportunity (and in some cases, the obligation) to correct errors made in earlier programs while preparing later ones. Errors resulting in grade deductions on one program still result in the same deductions unless corrected; this encourages students to continue working on problematic sections of code rather than simply "giving up" at the next due date.

Manageable Pieces

One of the difficulties with any large project such as this (especially given the constraints in our environment) is the need to divide a large project into several manageable yet meaningful pieces. If the pieces are too small (rarely a problem), students fail to sufficiently exercise the skills being taught. If the pieces are too large, students without good project management skills are prone to failure — and first-year students in CS2 may have little experience in project management.

The stages of this project seem to be of appropriate size. Program 1 is essentially a CS1 project and presents no new concepts; this allows students to begin working on the project from the first day of class. In particular, Program 1 gives students the chance to become familiar with the problem domain (and particularly the file format being used for input) in isolation from other topics. Program 1 also allows students to re-familiarize themselves with the Java language and environment (and gives students who did not take CS1 in Java one brief opportunity to catch up).

Each new program after the first allows the student to retain functional code from previous programs, while focusing their attention on new concepts (i.e., lists, trees, and graphics). Since the basic program framework has been established in previous programs, the new concepts can be considered in isolation.

In particular, Program 4, which introduces the Java AWT model, allows students to focus on building a graphical application independent of the underlying functionality. By the end of Program 3, students have constructed a fairly sophisticated database program with a significant amount of functionality which must be converted into graphical form; this gives students opportunity to work on a large graphical application independently of the functionality being supported.

Experience shows that the amount of work required seems to be appropriate; most students complete the work within the time required, and the few that miss the deadlines seem to have the usual excuses.

Project Variety

One of the benefits of this model is its flexibility. While the general notion of a database project has remained the same from term to term, there are an infinite number of database domains that can be used from term to term. This allows for re-use of the general framework without boring the instructor.

Even within the framework, there is considerable flexibility for variation. The exact types of dynamic data structures used in Programs 2 and 3 can (and have been) varied from term to term: e.g., singly versus doubly linked structures, multiply-sorted structures, and multiple levels of structures (e.g., lists of lists).

Both of the above features of the model seem to allow re-use of the general model without the danger of students re-using prior term projects (i.e. plagiarism) to complete their own projects. As always, the instructor must be careful to vary the projects sufficiently to make term-to-term plagiarism more difficult. Experience over the last several terms seems to indicate that this can be done successfully.

Additionally, the model is flexible enough to permit substitution of other related projects in addition to or in lieu of the specified projects. For example, in one term, students implemented a bookstore catalog, where book records included the number of pages and price of the respective book. As an exercise in recursive problem solving and backtracking, students were asked to solve a knapsack problem (using brute-force solving) using book page counts and prices from the bookstore database. This integrated the usually independent program on recursive backtracking into the rest of the program for the term. In another term, students implemented a course catalog, including information on course pre-requisites. In lieu of the final graphical program, students re-implemented the database as a directed graph and produced a topological sort of the courses in the database.

Project Constancy

Another benefit of the model is the relative constancy of the project. Offering essentially the same project over multiple terms allows the instructor to observe the problems which students encounter multiple times. In practice, students have similar problems with this project from term to term; over time, the instructor learns which problems are most likely and can be better prepared to counsel students when they encounter them.

The relative constancy of the model also makes the model easier to administer. An instructor can know what to expect in terms of the work required to introduce the model, counsel students, and score submissions, making time management easier, especially when one has other classes to teach.

Drawbacks

This model has been used over nine consecutive terms with success. Of course, this model has its drawbacks.

The constancy of this model was promoted above as an advantage; the same constancy can also be a drawback. It is a truism that "familiarity breeds contempt" [8]; excessive familiarity with the project holds the danger of taking shortcuts in the presentation or administration of the project. For example, preparing program descriptions from term to term naturally leads to copying previous program descriptions and making the appropriate changes (*e.g.*, changing the name of the database objects); it is all too easy in such a model to miss a necessary change.

The constancy of this model also lacks variety. The database project itself isn't terribly "nifty" [7] or uniquely interesting, although that can be alleviated somewhat by the choice of database in a given term.

Of course, one of the dangers of a constant model is the possibility of plagiarism as programs from previous terms may circulate among current students. It is another truism that "eternal vigilance is the price of liberty" [3]; the model requires attention to ensure that enough variety is introduced from term to term to make plagiarism more difficult.

The observant reader may have noticed that one significant topic, recursion, does not appear to be a major component of this project. Recursion is covered in depth in another stand-alone project involving recursive backtracking to solve a search problem (e.g., the "eight-queens" puzzle). It would be nice to incorporate this problem into the term-long project. (Of course, recursion is naturally integrated into discussions of dynamic data structures.)

CONCLUSION

We have presented a model for a term-long CS2 project which covers many of the traditional CS2 topics while requiring students to continually revise and extend previous code, thus providing students with practical incentive to write code in a modular fashion. The model is simple to present and administer, yet provides enough opportunity for variation to allow its use from term to term. We offer the model as an example of how to perform term-long projects even in environments in which such large projects might otherwise seem infeasible.

ACKNOWLEDGMENT

Thanks to Steven C. Cater for providing the basis for some of the ideas presented in this paper, and for critiquing early drafts.

REFERENCES

[1] Bartlett, J., *Familiar quotations*, 9th edition. Boston, Little, Brown and Company, 1901, http://www.columbia.edu/acis/bartleby/bartlett/

[2] Cater, S., "Modula-2 in a program design class," *Proceedings of the Second International Modula-2 Conference*, Loughborough, Leicesterschire, United Kingdom, October, 1991, pp. 292--301.

[3] Curran, J., *Speech upon the Right of Election*, 1790. As quoted in [1].

[4] Godfrey, M., and Grossman, D., "JDuck: Building a Software Engineering Tool in Java as a CS2 Project." In D. Joyce, ed., *Proceedings of the Thirtieth SIGSCE Technical Symposium on Computer Science Education (SIGCSE'99)*, pages 48-52. ACM Press, 1999.

[5] http://java.sun.com/products/OV_jdkProduct.html

[6] http://java.sun.com/products/jdk/awt/index.html

[7] Parlante, N., et. al., "Nifty Assignments Panel." In D. Joyce, ed., *Proceedings of the Thirtieth SIGSCE Technical Symposium on Computer Science Education (SIGCSE'99)*, pages 354-355. ACM Press, 1999.

[8] Publius, Maxim 640. As quoted in [1].

[9] Turner, J. A., and Zachary, J. L., "Using Course-Long Programming Projects in CS2." In D. Joyce, ed., *Proceedings of the Thirtieth SIGSCE Technical Symposium on Computer Science Education (SIGCSE'99)*, pages 43-47. ACM Press, 1999.