Are Case Studies a Good Teaching Tool for CS1?

Jacobo Carrasquel
January 1995
CMU-CS-95-132

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15123

This paper is a revision of an earlier paper used in the "Workshop On Case Studies for Advanced Placement Teachers" which took place in Clemson University, South Carolina, June 1993.

Abstract
In the summer of 1992 we decided to introduce the Case Studies, as described by Mike Clancy and Marcia Linn in their book Designing Pascal Solutions, in our introductory CS1 course for Computer Science majors. Our motivation was to do away with large programming assignments and to be able to teach what we consider important issues in a CS1 course. This paper will relate the experiences and results that we had with the use of Case Studies at Carnegie Mellon University during the 1992-1993 academic year. We decided to use a couple of old AP/CS tests to compare our students' performance against the results obtained by ETS.

Currently, the introductory courses at CMU are taught using C, not Pascal, and we are in the process of developing case studies which can be used with the current language. A new book written by Clancy and Linn is in the works.

Keywords: education, teaching, introductory programming, CS1, case study, Pascal, novice programmer.

Situation Prior to Case Studies

Before the adoption of the case studies, as discussed in Linn & Clancy, 1992, our CS1 course was a traditional one. Students taking this course are either CS majors or Engineers. The course uses a structure editor that runs on a Macintosh, called the Pascal Genie™ (Garlan & Miller, 1984; Miller & Chandhok, 1989). There are no prerequisites for this course, and approx. 30% of the students do not have any programming background.

The course was taught in Pascal over a 14-weeks semester in which the students met three times a week for 50 minute sessions. Each session had an instructor and a maximum of 24 students. There were no lectures in which all the students taking the course met. The students' grades were based on short assignments (labs), on-line quizzes and exams, and a mastery examination at the end of the term (Carrasquel, Goldenson, and Miller, 1985). The sessions (or recitations as we call them) took place in a computer lab equipped with a Mac IIsi for each student, a laser printer, a local network with access to a server, and a projection system to be used by the instructor.

As mentioned above, our course was not structured around lectures. We had a team of six full-time lecturers and four teaching assistants to manage the course. In general, one of the full-time lecturers (sometimes two) would be in charge of the planning and would function as the leader(s) for the entire course. The leader's main responsibility was to design the syllabus, sequence of topics, assignments, and grading policies. Each of the other lecturers and teaching assistants was in charge of two or three recitations.

First Semester of Case Studies

We decided to introduce case studies into our course during the Fall of 1992. Essentially, case studies describe the design process of an expert programmer: a statement of the problem, a solution, and a detailed explanation of the design and implementation made by the expert. The main motivation was to explore the advantages of the case studies as discussed in Linn & Clancy (1992): recycling (reusing code), divide-and-conquer (coding and testing complex programs one piece at a time), and alternative paths (alternative designs).

Another goal was to eliminate the need for large programming assignments. We found out that in general it takes too much time to explain the specifications of such programs and many times our students feel overwhelm by the complexity of the problem. Too much time is spent in dealing with design issues and decomposition of the problem.

Finally, we felt that it was a good idea to provide our students with good, interesting, and well-written programs. This way the students could have a model of what a good program is supposed to look like, something they could use as a model.

The six case studies that we adopted were: Check That Number, Banners With CLASS, The Calendar Shop, Roman Calculator Construction, You Are What You Eat, and Chess Challenges (Clancy & Linn, 1992). In addition to those six case studies, we designed and implemented a seventh one on dynamic data (linked lists). Each case study lasted two weeks (six sessions).

The approach during this first semester was to teach the course from within the case studies. By this I mean that we did not presented topics (language features) followed by a series of problems which required the student to apply specific tools. Instead, we we started each case study with an brief presentation of the problem followed by a discussion on specific language features that were required to solve the specific problem.

The first assignment consisted on reading the introduction of each case study. Then we discussed with the class the design of a solution. Once the design was solid, we moved to the implementation part which was done in modules. At the same time, we discussed most of the stop-and think and making sense questions included in each case study (Clancy & Linn, 1992). Each of the case studies used was divided into three to four labs. Most labs were based on the questions included in Clancy & Linn's textbook. Each lab had four to five of those questions, and in general, the student were given two days to complete them.

On several occasions, it was necessary to include more exercises that the ones suggested by Clancy & Linn. Borrowing a term from Mike Clancy, we called these “sand boxes”. The “sand boxes” were developed to introduced language features needed to solve a particular case study. The objective of the sand boxes was to allow the students to “play” within a small piece of code. For example, a “sand box” was a short program that had just a for loop. The students were asked to change the values of the loop and its direction. This way, the students could experiment at the same time that they were learning about this particular loop. In another “sand box,” students could modify the indexes and value of the elements in an array of characters. In general, the “sand boxes” were used at the beginning of each case study.

The students’ evaluation included three to four labs per case as explained above, two to three on-line quizzes, one multiple choice written test, and one on-line exam at the the end of each case study. The on-line quizzes consisted mostly of modifications to the labs written by the students. These quizzes lasted 10-15 minutes each. The multiple choice tests were taken from the AP/CS test given in past years. Each test had eight multiple choice questions and lasted about 10 minutes. The on-line exam was given at the end of each case, and it required either writing a short program similar to the case study or modifying the given solution. These exams lasted a full session (50 minutes).

In addition to these requirements, at the end of the semester, the students were required to take a mastery examination. A passing grade in this exam was required in order to pass the course (Carrasquel, Goldenson, and Miller, 1985). The exams used were constructed around the solutions to three case studies (Check That Number, Banners With CLASS, and Roman Calculator Construction). We asked the students to modify a randomly chosen original solution to one of these three cases.

Findings After the First Semester of Case Studies

One of the main difficulties that we encountered during the first semester was the preparation for each case study. We made the mistake of trying to teach Pascal within each case study. This did not work out well because the language constructs were needed before the students could design and implement their solutions.

For example, the first case study was “Check That Number”. In order to discuss a solution to this case study, it was necessary to talk about variables, types, assignment operator, basic input/output (write and readln), the math operators div and mod, relational operators, and the if statement. As a consequence, we found ourselves moving back and forth between the case study itself and what was needed, in terms of language, to understand it and solve it.

Another decision that was ill advised was not requiring a regular Pascal textbook that the students could rely on for readings or extra exercises.

In addition to the lack of a textbook, we tried to discuss and answer every one of the questions included in each case study (both stop-and think and making sense questions). Consequently, many times we ended up having very long (and sometime tedious) discussions which distracted us from the issues that the case study was supposed to address.

Finally, in order to justify the amount of time and work spent in the case studies, our mastery examinations were developed around three of them: Check That Number, Banners With CLASS, and Roman Calculator Construction. The problem with this decision was that we could not test everything that we wanted (or expected in a CS1 course) because we were restricted by the scope of each case study. Also, it was very difficult to create different exams with the same degree of difficulty and the same functionality.

Second Semester of Case Studies

One of the main problems we encountered the first semester was the lack of a regular Pascal textbook. So, during the second semester, we required, in addition to Clancy's book , Condensed Pascal by Cooper. This decision allowed us to assign readings before the material was introduced. The effect of this was that the students had, in advance, an idea of the language constructs that were going to be discussed in class. In addition, many students found it very useful to have a Pascal textbook (as opposed to having a manual). They could take a look at exercises and problems different from the ones used in our course. The adoption of a textbook was a definitely plus!

Another change that we introduced during our second semester was to reduce the number of case studies. This time we used four case studies (instead of six): Banners With CLASS, Roman Calculator Construction, You Are What You Eat, and Chess Challenges (Clancy & Linn, 1992), and our own case study on dynamic data (linked lists). Each case study lasted an average of three weeks (instead of two).

The teaching approach was different too. Rather than teaching from within each case study, as we did during the first semester, we decided to use the case study as a way to "wrap things up". Each case study was preceded by a series of lectures and short assignments where different language features determined by the case study were introduced. We spent an average of two weeks doing this; the third and final week was dedicated to the particular case study itself. So, each case study was used as an application of the tools learned instead of as a motivator.

The changes in the students' evaluation were as follows: we drastically reduced the number of stop-and think and making sense questions discussed in each case study, and most labs were not based on the questions included in Clancy & Linn's textbook. Also, in order to create some data to compare our performance, we gave a multiple choice quiz with each case study. Each quiz had eight questions that were taken from the 1984 and 1988 ETS AP/CS tests (the results are explained in the next session). We did continue to use the "sand-box" concept because this proved to be an excellent learning mechanism.

Finally, the mastery exams were not based on any of the case studies. We wanted to find out how well our students could apply all the programming principles introduced by the case studies and come up with a better (more fair and complete) pool of exams to be taken by the students.

Data

	Question	Year	ETS	CMU
Q1	#23	1988	57%	26%
Q2	#20	1984	47%	59%
Q3	#30	1988	82%	89%
Q4	#31	1988	30%	12%
Q5	#04	1984	84%	83%
Q6	#25	1984	41%	45%
Q7	#12	1984	70%	75%
Q8	#41	1984	14%	9%
Q9	#16	1988	52%	56%
Q10	#19	1984	67%	65%
Q11	#14	1984	57%	49%
Q12	#16	1984	76%	73%
Q13	#09	1988	79%	80%
Q14	#13	1988	52%	59%
Q15	#19	1988	51%	51%
Q16	#25	1988	33%	48%
Q17	#38	1988	95%	81%
Q18	#27	1988	27%	40%
Q19	#20	1988	58%	68%
Q20	#10	1988	76%	89%
Q21	#15	1984	83%	79%
Q22	#17	1988	62%	70%

Figure 1 - Questions used and percentages of correct answers in the quizzes

Figure 2 - This chart shows the percentage of correct answers in the 22 free-response questions that we used in our quizzes. (ftp file CMU-CS-95-132A)

Figure 3 - This chart compares the performance in the ETS quizzes and the total performance in the course. (ftp file CMU-CS-95-132B)

Conclusions

There are several conclusions to be drawn from our two semesters of experience using case studies. First, by using case studies, we managed to eliminate the need for large programming assignments written from scratch. The only time that the students were required to do so was during their mastery exam at the end of the course. After comparing the mastery grade distribution between the Fall of 1992/Spring of 1993 semesters (when we used case studies) and the previous 8 semesters, we found no noticeable differences in terms of student grades. In other words, based on our results, the lack of "large" programming assignments in our curriculum did not affect the students' performance in the mastery exam.

These results are important in the sense that a common complaint in CS1 courses is that many places do not have enough facilities (or the necessary time) required for long programming assignments. In our opinion, using case studies eliminated this need without affecting the students' learning.

The second conclusion is that based on the results from the multiple choice quizzes (taken from the free-response part of the 1984 & 1988 ETS AP/CS tests), we found no difference between our students' performance and that of those who took the AP/CS tests (refer to Figure 1 and Figure 3). This is relevant considering that most high school students take a full-year course before they take the ETS test, and in some cases, they are "coached" toward this type of test. There was nothing we did in order to prepare our students for these quizzes.

In Figure 1 there are only three questions in which our performance was very much different (more than 15% difference) from the ETS results. The first question was question 1 (question #23 from 1988). This question asked "If b is a Boolean variable, then the statement b := (b = FALSE) has what effect?" This question was given to our students after three weeks of classes. As you can see, only 26% of our students got the correct answer (compared to 57% for ETS). The second question was question 4 (question #31 from 1988). This question asked the students to trace some code in order to figure out the output produced by the program. The short program had call-by-value and call-by-reference procedures. Only 12% of our students (compared to 30% for ETS) answered it correctly. This second question was also part of the first quiz we gave our students. The third question was question 16 (question #25 from 1988). This question consisted of two nested loops swapping two elements of an array. The students had to determine the number of iterations of the inner loop. Forty eight percent of our students answered it correctly (compared to only 33% for ETS). This question was given after 9 weeks of classes.

Figure 2 illustrates the correlation between the normalized grade on the free-response quizzes and the course grade based on our own assignments. What we expected to see was a very strong correlation between these two (i.e. quiz performance and course work). As you can see, the correlation was not as strong as we expected. Nevertheless, this chart indicates that the students who did well in the quizzes also did well in our assignments.

In conclusion, we found the case studies a valuable teaching tool and recommend that they be incorporated into CS1. Using case studies, we were able to reduced the need for large programming assignments without hurting our students' performance. Also, the case studies allowed us toexplore some of the "eight principles of program design" as discussed by Linn & Clancy (1992a, 1992b), specifically: (2) ability to reuse templates, (3) design of alternative solutions to a problem, (6) incremental development of the code, and (8) writing self-documenting code.

References

Carrasquel, J., Goldenson, D. & Miller, P. L. (1985). Competency Testing in Introductory Computer Science: The Mastery Examination at Carnegie Mellon University. SIGCSE Bulletin, 17 (1), p.240. Abstract, full paper available on request.

Clancy, M.J. & Linn, M.C. (1992). Designing Pascal Solutions: A Case Study Approach. New York: W. H. Freeman and Company (Computer Science Press).

Garlan, D. & Miller P.L. (1984, April). GNOME: An Introductory Programming Environment Based on a Family of Structure Editors. SIGSOFT/SIGPLAN Bulletin.

Linn, M. C. & Clancy, M. J. (1992a). Can experts' explanations help students develop program design skills? International Journal of Man-Machine Studies 36, pp. 511-551.

Linn, M. C. & Clancy, M. J. (1992b, March). The Case for Case Studies of Programming Problems. Communications of the ACM, 35 (3) pp. 121-132.

Miller P.L. & Chandhok, R. (1989). The Design and Implementation of the Pascal Genie. ACM Computer Science Conference, Louisville, Kentucky. February,1989.