2006-1027: SOFTWARE ENGINEERING PRACTICES USED FOR RETENTION?

Joseph Clifton, University of Wisconsin-Platteville

Joseph M. Clifton is a Professor in the Department of Computer Science and Software Engineering at the University of Wisconsin – Platteville. He has a Ph.D. from Iowa State University. His interests include software engineering, real-time embedded systems, and software engineering education.

Software Engineering Practices Used For Retention?

Abstract

Software engineering practices have been integrated into many computing curricula in a variety of ways. Authors have written about integrating software testing throughout their curriculum and using software development methods such as Extreme Programming[2, 3, 4]. Papers have been written on how some software engineering techniques, such as pair programming, can help increase retention, particularly of female students[7, 8].

This paper suggests that other software engineering practices can be used to help increase the success rates in lower division courses, which should translate into increased retention rates. In particular, use of detailed work plans and periodically monitored time logs and version control check-ins is examined. The underlying assumption is that students need to be encouraged to start programs early and work steadily towards completion.

Introduction

Over the past ten years, about twenty-five universities in the US have established undergraduate software engineering programs. Moreover, software engineering has worked its way into a large number of computer science and computer engineering curricula. Indeed, the Computing Curricula 2001 for Computer Science lists software engineering as one of the fourteen areas comprising the CS body of knowledge[1]. There are eight units in software engineering designated as core with a recommended minimum coverage of thirty-one hours.

Over the years, many software engineering and computer science educators have advocated integrating software engineering practices into the computer science curriculum. For example, authors have written about integrating software testing throughout their curriculum and using software development methods such as Extreme Programming[2, 3, 4]. Recently, some authors have investigated whether certain software engineering techniques, such as pair programming, can be used to increase success rates in CS1 courses and retention of women in CS-related majors[7, 8].

In this paper, we investigate whether other software engineering practices can be used to increase the success rate in the introductory courses and thereby help increase retention. In particular, we look at requiring specific work plans together with periodically monitoring time logs and checks-ins to version control.

Background

The University of Wisconsin, Platteville is a relatively small undergraduate university with about 6000 students. It has offered a BS degree in Computer Science since 1982 and a BS degree in Software Engineering since 1999. The CS1 and CS2 courses offered prior to the fall of 2001 were three-credit semester courses.

In the fall of 1995, the language used in the CS1 and CS2 courses was switched from Pascal to C++. Some reasons for the switch given at that time were the need to go to an object-oriented language in CS2 and the desire to choose a language that would "look good to industry", since most of our computer science graduates went to work for industry immediately after graduation. Presumably, at least in part due to the complexity of the C++ language, the success rates in the CS1 and CS2 courses plummeted and the department searched for ways to improve them. A two hour lecture, two hour lab format was tried in CS1 with marginal improvement. A change to a simpler development environment that supported only a subset of procedural C++ and a three hour lecture, one hour lab format brought some improvement to the CS1 success rates. It should be noted that the CS1 course remained at three credits, so this extended the contact time beyond what was considered normal for our university.

In the fall of 1999, the language used in CS2 was switched from C++ to Java. Java offered a simpler environment to learn object-oriented programming. Furthermore, it was felt that the transition from procedural C++ to Java wouldn't be significantly different than the transition from procedural C++ to object-oriented C++ and any difficulties in the transition would more than be made up for by not having to cover material such as pointers, memory management, operator overloading, and templates. Despite this, success rates still remained unacceptably low.

In the fall of 2001, the CS2 course was split into two three-credit courses and these courses were cross-listed in both computer science and software engineering, since they were required for both majors. In this paper, the two resulting courses will be referred to as CSSE2-I and CSSE2-II. There were several reasons for the split. One reason was an attempt to increase the success rate, which was still below 50% despite earlier attempts at improvement. Another reason was to reintroduce C++ material that had been dropped from the curriculum. Yet another reason was to incorporate more software engineering practices by having students:

- write design documents, test plans, and test specifications
- participate in reviews and inspections
- use version control
- do some programs/projects in teams

Although the software engineering additions were primarily in support of the newly-created software engineering program, it was felt that these changes would also benefit the computer science students, since the majority of the computer science students took software development jobs after graduation.

The resulting course structure is that students start in CS1 where procedural programming using C++ is taught. In the CSSE2-I course, object-oriented programming is taught and the language switches to Java, for reasons specified above. The CSSE2-II course is taught in C++, incorporating topics such as memory management, operator overloading, and templates. The introduction of data structures is split between the two courses. The CSSE2-I course covers stacks, queues, linked lists, and an introduction to algorithm analysis. The CSSE2-II course redoes linked lists using pointers and covers trees, graphs, recursion, and goes a little deeper into

algorithm analysis. Very little additional depth or breadth of data structures was added due to the split.

The expectation was that the overall success rate would be higher since CSSE2-I would be a gentler introduction to the material and the harder concepts in CSSE2-II wouldn't be introduced until the students had success with objects and some simple data structures. To our surprise, the percentage that completed CSSE2-II the first time it was taught in the fall of 2001 was still less than 50%. After some modifications, the success rate improved somewhat, but still averaged less than 60% for the next three semesters.

Students who dropped or flunked were informally questioned about why they thought they didn't succeed in the course. The most common answer given was failure to make the transition from Java to C++ quickly enough. When questioned further, most admitted to waiting too long before starting the programs.

Approach

The assumption was made that students needed to be encouraged to start programs early and work steadily towards completion. Giving a student a program and a deadline for completion was insufficient. Guidance and justification were taken from the real world. Software developers are rarely handed a project and told to return a specified number of months later when they have it completed. Management often requires plans, frequent progress reports, etc. So the inspiration was to attempt to incorporate some of these types of activities in the hopes of improving the success rate of students in the CSSE2-II course and ultimately improve the retention rate. It was expected that an additional benefit would be that students would start thinking about the process that they use to produce software and how they could start to recognize, understand, and improve their personal software process.

Other authors have discussed the use of some of these techniques. For example, some have discussed incorporating the Personal Software Process (PSP) techniques of time estimation and time logging into the CS1 and CS2 courses[5]. Others have discussed using time logs and Gantt charts in the CS1 course[6]. The motivation in both instances was to instill good personal software process practices early in the curriculum. We have the added motivation of wanting to increase the student success rate.

Starting in the fall of 2003, we employed the following practices in the CSSE2-II course in addition to those listed in the Background section above:

- Specific work plans
- Periodically monitored time logs
- Periodically monitored version control check-ins

The CSSE2-II course has five assigned programs, with approximately three weeks allocated for each. For each program, a work plan must be submitted that specifies how the student plans to

complete the program by the due date. These must list dates and times that the student will work on the program together with what they plan to accomplish for each specified time period. It is not enough to put down estimates for broad categories such as design or coding. They must be specific, such as: 9/15: 4:30 – 5:30, Test the Stack Class. Furthermore, the work plan must demonstrate that the student has a good understanding of the program specification. Students are usually given only a few days after a program description is made available to submit their plans. Students failing to turn in a plan are penalized approximately 20% of the total grade of the program. Those turning in insufficient plans are required to redo them.

For the first program, the students are required to bring their work plans to the instructor for critique. This gives some individual attention to each student very early in the course and gives the instructor time to assure the student that these exercises are meant to help the student. Work plans for the remainder of the programs are submitted electronically. For programs/projects involving groups, there is an added requirement that the students also specify the place that they will meet for each time period.

Requiring such detail has two objectives:

- Encourage students to read, understand, and think about what needs to be done - very early
- Assure that students plan specific times they will allocate to the program

For time logging and version control, each student is allocated individual workspace on a Novell share drive. Using homegrown administrative tools, the workspaces are set up so that only the student (or group members) and the instructor can access the space. Visual SourceSafe databases are also created in this workspace. It should be noted that Subversion on Linux will probably be used the next time the course is taught. One reason for switching would be to increase accessibility for those students off campus. Another reason is related to server and throughput problems for the Novell share drives this past year which have caused an undesirable amount of downtime.

Students are expected to keep an up-to-date time log in their workspace. Spot checks are performed every two or three days. This frequency assures steady work but also takes into account that students and instructors have other obligations. A student is penalized one point every time that a spot check reveals that their time log is more than three days old. Likewise, students are required to periodically check their source code into version control. This is spot-checked every few days after allowing some time for design. A student who fails to show some non-trivial progress for more than a three day period is penalized one point, unless, of course, the student has completed a final submittal of the program.

All work plans are checked carefully to assure that students intend to meet these criteria. It should be noted that a grace date is provided for each program. Although the student must plan to meet the due date, they still receive full credit if they complete the program by the grace date, which is generally about three days after the due date. This was also taken from industry, where

planned completion dates are often "moved up" from the actual due date to allow for the inevitable unforeseen problems.

Other software engineering related deliverables such as test beds, test plans, and design documents are also required for some programs/projects. However, since these are not periodically checked, they won't be further discussed in this paper

Results

As noted previously, the additional practices listed in the section above were first tried in the CSSE2-II course in the fall of 2003. Results from the first trial were very encouraging: over a 90% successful completion rate. We believed that we had hit upon a magic formula! The subsequent semesters' results were less gratifying, but were still an improvement over the results from previous semesters.

The percentages of students in CSSE2-II earning each grade from the past nine semesters are shown in Table 1. Figure 1 shows the same data as a stacked bar chart. The last five semesters incorporated the new techniques. It should be pointed out that the instructors varied from semester to semester, but all have a history of similar grading styles for the different courses that they teach.

Term	A	B	C	D	F	W	C or Better	# Students
F01	12.2%	12.2%	24.5%	4.1%	22.5%	24.5%	49.0%	49
S02	8.0%	20.0%	24.0%	8.0%	40.0%	0.0%	52.0%	25
F02	17.7%	29.4%	17.7%	8.8%	17.7%	8.8%	64.7%	34
S03	18.8%	28.1%	9.4%	9.4%	31.3%	3.1%	56.3%	32
F03	12.5%	58.3%	20.8%	0.0%	0.0%	8.3%	91.7%	24
S04	13.0%	47.8%	17.4%	0.0%	4.4%	17.4%	78.3%	23
F04	11.0%	40.7%	29.6%	3.7%	0.0%	14.8%	81.4%	27
S05	6.3%	31.3%	25.0%	25.0%	6.3%	6.3%	62.5%	16
F05	5.6%	44.4%	11.1%	11.1%	11.1%	16.7%	61.1%	18

Table 1 - CSSE2-II Grades by Semester

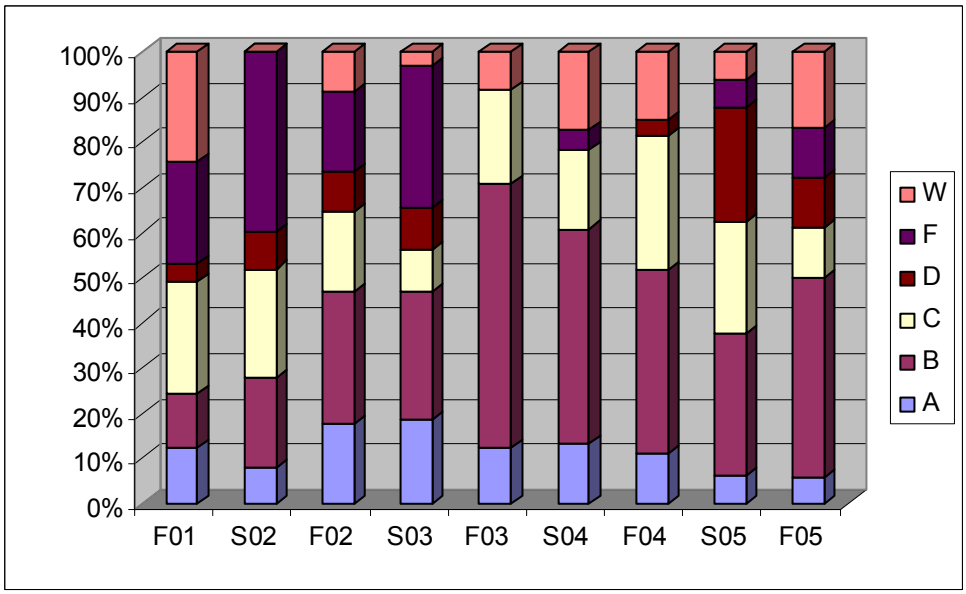

Figure 1 - CSSE2-II Grades by Semester

Assuming two sets of data samples, one for the first four semesters and one for the last five semesters, a t-test can be used to determine if there is a statistical difference between the average percentages of students successfully completing the course. One might first attempt to argue that it should be a one tailed test, since it is expected that the percentage of successful completions for the second set should be at least as high as the first set. But this would fail to take into consideration that some students may drop just because of the extra work. As for the question of whether or not the population variances should be assumed to be equal, we could not think of any convincing arguments why they shouldn't be assumed to be equal.

Table 2 shows the results of using Microsoft Excel to run a two-sample t-test assuming equal variances at the 5% significance level. Variable 1 is the success rate for the first four semesters. It is obtained by using the first four rows in the "C or Better" column from Table 1. Variable 2 is similarly obtained, except using the last five rows.

The two-tailed test rejects the hypothesis that the means are equal (P=0.031), thus the claim that these methods help improve the success rate can be made from the current data and assumptions. As a precaution, the test was also run assuming unequal variances and resulted in P=0.028, a slightly stronger statistical argument for the claim.

t-Test: Two-Sample Assuming Equal Variances		
	Variable 1	*Variable 2*
Mean	0.55485	0.74982296
Variance	0.00467639	0.01694949
Observations	4	5

Pooled Variance	0.01168959	
Hypothesized Mean Difference	0	
df	7	
t Stat	-2.6882425	
P(T<=t) one-tail	0.01558248	
t Critical one-tail	1.8945786	
P(T<=t) two-tail	0.03116495	
t Critical two-tail	2.36462425	

Table 2 - Results of t-test of Average Success Rates

In addition to the above analysis, there were some positive qualitative results. Many students stated that they thought that the planning and monitoring did help them. They said that they had procrastination problems in earlier courses and these practices helped them stay on task. Furthermore, many stated that they acquired a better understanding of their personal software process. There were some, however, that felt the extra work required was not worth the time spent. As one may or may not expect, those saying this typically were the "A" students, who easily met program deadlines without being monitored.

The next question becomes whether this increase in success rate translates into an increased retention rate. One might expect that the answer would obviously be yes, but attempting to quantify the amount is more difficult. A study of the data from fall 2001 through spring 2004 shows that only about 25% of the computer science and software engineering majors who didn't satisfactorily complete CSSE2-II in any given semester eventually chose another major, dropped out of college, or transferred to another college. The other 75% made it through the course on the second or third time, and stayed with the major. Of the 75%, about 85% of those made it through the second time and the remainder made it through on the third time. This compares to about 10% of those who did successfully complete the CSSE2-II course but also eventually chose not to continue as a computer science or software engineering major. Therefore, the tie between success rate and retention is not as high as we had expected. Perhaps by the time the students get to the CSSE2-II course, most have made a commitment to their major.

A couple of items from Table 1 that require further discussion are the declining number of students enrolling in CSSE2-II and the lower success rates the last two semesters. The former is addressed in the next section. As for the latter, the instructors from the last two semesters admitted that due to time constraints, the efforts of assuring student adherence were not as good as they were the previous semesters. It should be noted that the department currently does not use student graders, so the instructor is responsible for the extra work of periodic monitoring and additional grading.

Next Steps

In 2003, concerns over the success rates in CSSE2-II were the motivation to come up with ways to try to "force" students to start programs earlier, since it was believed that procrastination was a

big cause of failure in the course. At that time, the enrollment in the computer science and software engineering programs was still fairly solid; however, there was concern that the enrollment in the upper division courses could start to suffer.

Since that time, the enrollment in our computer science and software engineering programs has continued to decline. Retention and success rates in the CS1 and CSSE2-I courses have become major concerns. Furthermore, recruitment has become an issue. Some national articles attempt to explain the decline in students choosing computer-related fields, but that discussion is beyond the scope of this paper. However, one potential factor, the perception of large failure rates in computer science and software engineering courses, is related to this paper since it is related to curriculum and pedagogy. A number of steps are being proposed, two of which are: 1) Use the ideas presented above in the CSSE2-I course and 2) Use pair programming in the CS1 course.

In the fall of 2005, the CS1 instructors experimented with some use of pair programming in CS1. There were differences in how each approached it. Furthermore, there are still debates about when to introduce pair programming, early in the semester or later, and how many programs and laboratories should use pair programming versus done individually. Despite this, the success rate for CS1 for fall 2005 was 74% compared to about 65% the previous three fall semesters.

Also in the fall of 2005, the techniques listed in this paper were added to CSSE2-I, with the exception of using version control. Although there are not enough data to do a statistical analysis, Table 3 shows the percent of students earning each grade from the past nine semesters. As can be seen, the success rate for the fall of 2005 appears significantly better than the success rates from past semesters. Furthermore, a study of the data from fall 2001 through spring 2004 shows that about 50% of the computer science and software engineering majors who didn't satisfactorily complete CSSE2-I in any given semester eventually chose another major, dropped out of college, or transferred to another college. So achieving higher success rates in the CSSE2-I course should better translate to higher retention rates than it does for CSSE2-II.

Term	A	B	C	D	F	W	C or Better	# Students
F01	16.7%	20.8%	20.8%	6.3%	6.3%	29.2%	58.3%	48
S02	17.5%	25.4%	25.4%	6.3%	6.3%	19.0%	68.3%	63
F02	18.5%	22.2%	16.7%	3.7%	9.3%	29.6%	57.4%	54
S03	8.1%	16.1%	22.6%	8.1%	24.2%	21.0%	46.8%	62
F03	12.5%	22.5%	15.0%	12.5%	15.0%	22.5%	50.0%	40
S04	8.0%	16.0%	42.0%	10.0%	6.0%	18.0%	66.0%	50
F04	12.9%	32.3%	19.4%	3.2%	12.9%	19.4%	64.5%	31
S05	5.4%	32.4%	21.6%	8.1%	10.8%	21.6%	59.5%	37
F05	11.8%	35.3%	32.4%	2.9%	5.9%	11.8%	79.4%	34

Table 3 - CSSE2-I Grades by Semester

Another step the department must take is to address the issue of the extra work of periodic monitoring and additional grading. Some colleagues question whether the extra work required of the instructor is worth it. It would be helpful if tools were available to help automate the monitoring; however, for the techniques to be successful long term, the department needs to hire student graders to do this work. One hurdle is getting administration to agree to the additional funding.

Conclusion

Use of software engineering practices holds potential for increasing the success rates in introductory computer courses and subsequently increasing the retention rate. Some authors have looked at techniques such as pair programming as holding potential for increasing retention. This paper presents evidence that other techniques also show promise, such as specific work plans and periodically monitored time logs and version control check-ins. The statistics show an increase in the success rate can be achieved using these techniques. The tie to increased retention is a little harder to prove. There is some evidence that increasing the success rate in the CSSE2-II course can achieve increased retention; however, increasing the success rate in lower level-level courses, such as CSSE2-I, appears to hold more promise.

We believe the software practices in and of themselves are important for the student to experience. One concern is how to address the extra time and effort required for the periodic monitoring and additional grading.

References

[1] *Computing Curricula 2001: Computer Science,* December, 2001. Online [March 4, 2005]. Available WWW: http://www.acm.org/education/curricula.html or http://www.sigcse.org/cc2001/

[2] Goldwasser, M. A Gimmick to Integrate Software Testing Throughout the Curriculum. *The Proceedings of the Thirty-third SIGCSE Technical Symposium on Computer Science Education* (2002), 271-275.

[3] Hazzan, O. and Dubinsky, Y., Teaching Software Development Methods: The Case of Extreme Programming. *The Proceedings of the Sixteenth Conference on Software Engineering Education* (2003), 176-184.

[4] Hilburn, Thomas B., Software Engineering – from the Beginning. *The Proceedings of the Ninth Conference on Software Engineering Education* (1996), 29-39.

[5] Hilburn, T. and Towhidnejad, M. Doing Quality Work: The Role of Software Process Definition in the Computer Science Curriculum. *The Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education* (1997), 277-281.

[6] Hou, L. and Tomayko, J. Applying the Personal Software Process in CS1: An Experiment. *The Proceedings of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education* (1998), 322-325.

[7] Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., and Balik, S. Improving the CS1 Experience with Pair Programming. *The Proceedings of the Thirty-fourth SIGCSE Technical Symposium on Computer Science Education* (2003), 359-362.

[8] Werner, L., Hanks, B., McDowell, C., Bullock, H. and Fernald, J. Want to Increase Retention of Your Female Students? *Computing Research News,* Volume 17, Number 4, March 2005.