Fighting the Student Dropout Rate with an Incremental Programming Assignment

Tuukka Ahoniemi **Essi Lahtinen** **Teemu Erkkola**

Institute of Software Systems
Tampere University of Technology,
PO Box 553, Tampere, Finland
{tuukka.ahoniemi, essi.lahtinen, teemu.erkkola}@tut.fi

Abstract

Large programming assignments can become huge obstacles to novice programmers, especially as teachers usually lack the resources to guide students sufficiently in-depth for the whole time. Changing the assignment to an incremental one consisting of smaller phases built on top of one another helps students to start in time, stay in time, and avoid succumbing to the huge workload.

We made the large assignment in a programming course incremental and got positive results when measuring the students' submission behaviour and their opinions on the phasing. The students felt that they were aided instead of just given more deadlines. The students willingly took advantage of our approach and really appreciated it. This article explains how we made our incremental assignment, how students used the phases, and how they felt about them.

Keywords: Novice programmers, programming education

1 Introduction

The bigger the programming assignments, the harder it is for students to grasp them. This is partly because novice programmers have difficulties in identifying the big picture and they approach the program line by line (Robins et al. 2003, Soloway & Spohrer 1989). The students may know the required programming concepts, but lack the skills to apply them (Lahtinen et al. 2005, Winslow 1996).

Novice students often also lack experience on the overall programming process, as they have only had experience in writing code fragments. They do not acknowledge the time required for proper testing and debugging, nor possibly even how to test or debug at all (Soloway & Spohrer 1989).

Teaching programming in general can be seen as an incremental process (Robins et al. 2003) — in a way it is natural to base new information on top of old. Some industrial programming schemes implement this idea in a systematic incremental approach.

In response to the known problems we decided to apply a similar incremental approach in a novice programming course that has a rather large programming assignment. We also wanted to study its effects on the dropout rate, the students' behaviour in applying the somewhat optional phases, and their perception of the incremental assignment and its workload. To address these questions, we conducted surveys during the course.

In this paper we describe the problems encountered in our introductory programming course related to its relatively large programming assignment, and our solution of an incremental assignment. Then we introduce the ways we have measured the effectiveness of our experiment and present both the quantitative and qualitative results. The results are then discussed and conclusions are drawn.

Copyright ©2008, Australian Computer Society, Inc. This paper appeared at the *Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007)*, Koli National Park, Finland, November 15-18, 2007. Conferences in Research and Practice in Information Technology, Vol. 88. Raymond Lister and Simon, Eds. Reproduction for academic, not-for-profit purposes permitted provided this text is included.

2 The Course Setting

Our target course is the second introductory-level course in programming at Tampere University of Technology held in the spring semester for first-year students. After the course, which involves a few small programming assignments and one large one, students should know the very basics of object-oriented programming using C++.

2.1 The problems of the large programming assignment

The large programming assignment in our course tests the skills learnt through the whole course. It is large so that students can see the real benefit of using classes, get a grasp of the whole software process, and gain programming experience. The assignment should require intensive work over at least a couple of weeks, with careful design and testing also involved. The assignment is evaluated with automatic assessment, and when it is sufficiently correct, a teaching assistant manually gives it a summative grade.

Despite our explanations and urging, students have had a tendency to ignore the large size of the assignment. We have identified the following problems caused by this:

- Starting way too late
- A very hastily done implementation which merely *works*, but the student thinks it is perfect
- A very hastily done implementation which the student knows not to be adequate, but says he/she lacked the time to make it better
- A big dropout rate in the beginning (they don't know where to start) and in the last few days (they cannot get it working)
- Multiple desperate student requests for help in the last days before the deadline
- Many 'extratimers' (we have had a policy of giving few days' extra time with a grading penalty)
- Much negative feedback on the assignment size

We offer constant assistance throughout the course. As the students with most problems have usually started working too late, they do not take advantage of this support early enough.

To ease the start and guide the students to implement correctly designed solutions, the assignment has always begun with a *design phase*. Before they start programming the students have designed the classes/modules thoroughly, receiving feedback from a teaching assistant. After the design phase the students have had at least six weeks to implement the program. This has resulted in the problem that not all students have understood the need to be working bit by bit for the whole six weeks.

2.2 Incremental Assignment in Spring Semester 2007

To tackle the big problems we have faced with our assignment, we decided to apply the idea of building the program incrementally. The design phase was left as is, but the actual implementation part was divided into four incremental phases and a final submission that corresponded to the only submission of previous years.

The assignment description was released earlier, as a whole, describing what was to be done during the whole assignment. We also provided an additional phase submission specification which described what was to be implemented in each phase.

We designed the assignment to be easily divided into phases so that the next phase would require a working implementation of the earlier parts. The phases were still separate enough to be tested and debugged separately. As we wanted students to find the phases helpful rather than just adding further deadlines, we ruled the phases compulsory to the extent that skipping two phases in a row was not allowed (meaning that at least two phases were obligatory).

To encourage the students and to monitor their progress we gave them the opportunity to use automatic assessment for the phases. The requirements for these automated assessments were not as strict as those of the final submission, but missing a phase would give a student the extra responsibility of thoroughly validating the correctness of that phase. The different phases are explained in more depth in the following.

2.2.1 The phases

As in previous years, the task began with the design phase.

The first and second implementation phases held nothing actually new for the students. The first phase was only the start of the program, that is, the parsing and validating of the given program arguments. The second phase was to implement a completely functional command shell for the program so that it would understand which of the commands (and their arguments) are legal and which are not. We wanted the students to have a working command shell at this point to provide the means for testing the further functionality of the program.

The third implementation phase had the first tricky elements of the assignment. The main focus of this phase was on configuration file parsing. A completely correct file parser required precision and labour, and was supposed to construct objects of the used classes. The classes did not need to be fully implemented in this phase but the initialization and output functions were required to work correctly.

The fourth implementation phase was basically to complete the program. Though it may seem that this would be the largest phase, this was actually the most convenient one. If the third phase worked (the classes were instantiated correctly) all that remained to students was the implementation of the rest of the methods — which they had already designed in the design phase.

The final submission had no more functionality than the fourth implementation phase. The automatic assessment tool was now testing each part of the complete program, which was also graded manually by a teaching assistant. The final submission also included also a documentation of the program. Only the design phase and the final submission contributed to the grade for the assignment.

3 Measuring the Effectiveness

To measure the effectiveness of the approach we collected data on all student submissions, presented a short survey after each phase, and at final submission time collected open feedback about the assignment as a whole.

When a student had passed the automatic assessment on a phase, the system asked the student the following multiple-choice questions:

- How easily did you manage through this phase?
- How clear is the implementation of the next phase for you right now?

4 Results

In this section we present results on the effects of our approach: quantitative results from submission statistics and student questionnaires and qualitative data from student feedback.

4.1 Submission statistics

The data collected from submissions give the total number of submissions, which is compared to the data from the previous year. For this year we have also derived 'phase paths' which show more precisely how students completed the phases. The average times of the submissions in each phase are also presented.

The dropout statistics

For measuring the student dropout rate during the large programming assignment (not during the whole course) we compared the number of students who attempted the design phase with the number who made final submissions. We also calculated the number of extratimers (students who submitted late). The results are shown in Table 1.

Table 1: Dropout statistics from 2006 and 2007

	2006	2007
Students in the design phase	240	196
Final submissions	183	143
of which extratimers	73	6
Percentage of dropouts	23.7%	27.0%
Percentage of extratimers	39.9%	4.2%

The results show that the plain dropout rate was not reduced at all, but actually increased slightly. However, the number of students who submitted late not only decreased but fell almost to nothing.

The 'phase paths' taken by students were analyzed in order to distinguish different student behaviours and how successful those behaviours were. Over 75%

of the students completed the first phase, which is already a great improvement, because it means that most of the students began implementing their program on time. Over 80% of these students passed the final submission. Only 47% of those who did not pass the first phase passed the final submission.

The second phase was seemingly as easy as the first one, as an almost equal number of students completed it. However, 83% of students who completed this phase, regardless of how they did in the first phase, went on to complete the entire assignment. This clearly supports our assumption that a phased exercise generally makes students start their work earlier, and that those who do so are much more likely to finish the work they have already started.

The third phase was without a doubt the hardest phase, as only 27% of all students submitted a work that passed it. Of these 27%, 64% passed every phase and the final submission.

The most popular paths through the phases were to complete every phase but the third (25.9% of the students) and to complete every phase (17.1% of the students). Surprisingly, the path that seemed to be of least work — completing only every other phase — attracted only 7.3% of students.

The more the students completed phases, the more likely they were to do the next phase. This is best seen in the case of the third phase. Roughly half the students who had completed both first and second phases also completed the third phase. Of those who had completed only the first phase, a little over one third completed the third phase. Of those students who had completed the second phase but not the first, only about 11% completed the third.

Times of the submissions

In addition to the lower number of extratimers, students generally submitted their work earlier than usual. Table 2 shows that although the first phases were easier, students still clearly began working on them earlier than the last few days before deadline. Even the hardest phase — the third — was on average submitted more than 30 hours before the deadline. In comparison, the non-phased assignment of 2006 had a submission time average of 25 hours before deadline.

Table 2: Students' average submission times (hours before deadline)

Phase 1	Phase 2	Phase 3	Phase 4	Final
78	55	32	46	67

4.2 Student feedback

Recognizing the hard part

As seen in Figure 1, after the first phase students generally knew what they were supposed to do next. Some of them still had not grasped the entire exercise description at this point and thus were not sure if they understood it correctly. After the second phase they realized that they had not done anything as big as the third phase before, and this is reflected in the greater uncertainty in the figure. However, after the third phase most of the students had a very clear idea of how to continue. These results are along the same lines as the results shown by the phase paths in terms of the difficulty of the phases, and reflect directly on the smaller number of submissions in the third phase.

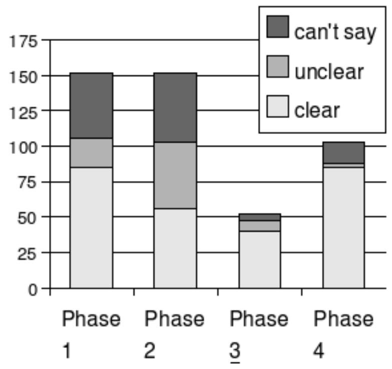

Figure 1: Students' answers to the question "How clear is the implementation of the next phase for you right now?" after each phase

Effect on students' workload

Only 7% of the students thought that phasing increased the workload "a lot more" or "some more". In general the students didn't feel that their workload increased much — 21% thought that it increased a little — whereas 40% of the students experienced the phasing as guidance, hence lowering the amount of work required.

Open feedback answers

We had 124 final surveys to analyze. Of these, 51 made no reference to the incremental assignment, leaving 73 that made some mention of the phasing. The results are shown in Table 3.

The open feedback results clearly show that the experience was really positive for the students. They felt that the phases were a huge aid for them in forcing them to start early enough with a good schedule and in helping to divide the assignment into reasonable parts. Most of the student complaints were about the uneven sizes of the phases — the third phase was surprisingly large. None of these students concluded that the incremental assignment was a bad thing.

5 Discussion

Increasing the odds to success

An early start to the work seems to be a strong indicator for passing the assignment. Compared to our old system, 'starting early' means to start with phase 1 instead of waiting until phase 2. This seemed to be more attractive than to 'start working weeks earlier on a huge amount of work'. The students also started working early on the subsequent phases. One reason for this might be that the students have a clearer picture of what they are doing so it is easy to start well before each phase deadline. Anyway, some sort of change in attitude appears to have happened, since there were far fewer students who submitted in extra time.

Missing the first (or any) phase worked as a concrete message to students that their work was not on schedule. Because they had slacked off during the first phase, they now had to double-time to catch the others during the second phase. So although no actual penalty was given for missing a phase, the sheer need to work harder and faster may have done the job. We also emphasized in the lectures that these phases provide a schedule that we find realistic — in our opinion, missing a phase means you are late.

By phasing the exercise we were able to isolate the

Table 3: Results from the open feedback (n=73)

Overall evaluation on the incremental assignment mentioned really good	51
Overall evaluation on the incremental assignment mentioned fine / ok	4
Overall evaluation on the incremental assignment mentioned bad	0
Thanked for forcing to begin early and/or providing a ready schedule	42
...and in addition thought that would not have survived without the forcing	8
Thanked for providing ready-made division of the assignment	11
Did not like the ready-made division	1
Thanked for the ability to automatically test program in each phase	7
Complained about the uneven sizes of the phases	16
Complained about the schedule of the phases	8

harder part and let the students handle it separately. After the hard part the students knew how to finish the assignment. On the other hand, before the difficult phase they had already completed half of the assignment. Quitting at that point would be a pity.

One of the key aspects to a phased exercise is the way a student views it. As students tend to pick the path of least work, the success of phasing can be determined by how students take advantage of the system. In our case a very small percentage of students did only the minimum required submissions. As students themselves wrote in open feedback, phasing was for their help and they were pleased to take advantage of it.

Some of the students said that phasing actually increased their workload. This might be because, as many students confessed in the open feedback, they completed some of the phase submissions in such a rush that their result was not suitable as a basis for building anything new. Thus they had to refactor major parts of that phase for the next phase, resulting in excess work just because of additional deadlines. What these students did not state (and probably did not know) was that even without the additional deadlines they would probably have made many of the same mistakes, and would thus still have had to fix them before proceeding. Luckily, many of these students still recognized that they had at least learned valuable information when doing these refactorings.

Guiding or restricting?

A single student pointed out that phasing spoiled the independent design of the program, forcing it into a single mould. However, the imposed structure did not differ greatly between the phased exercise of 2006-2007 and the non-phased exercise of 2005-2006. The actual assignment description was written in a similar way in both years. The phase description part of the 2007 assignment consisted mainly of a list of functionalities required for each phase — exactly the same things as in the assignment description — and some phase-specific output strings. The phases were also quite natural independent parts of the main assignment description. Therefore the only significant 'forced' parts of the phasing were starting early and writing the program in a smart, easily testable way. Students were given the option of making their own schedule, so this part of the assignment was in no way forced. The student might also have meant the elaborate guidance of the design phase. Good designs meant following the course teachings in regard to programming style and forbidden methods; to accept poor designs would have resulted in bad programming style, greater dropout rates because of bad design, and even more whining and complaints. Because of this, students with bad designs in the design phase were guided toward the intended solution for their own good.

6 Conclusions

Despite putting in a lot of effort designing and implementing an incremental programming assignment, we did not manage to lower the actual dropout rate during the assignment. However, we did see a dramatic reduction in the number of students submitting their work in extra time. The students started and finished early, and thus with better outcomes. Most students who dropped out did so during the first two phases. Thus there were fewer frustrated students who had worked for weeks without completing the assignment.

The students' opinions on how the incremental assignment affected their workload were also encouraging. Only a minority perceived that the phases increased the workload, and many students thought that the guidance decreased their working hours. Overall the feedback about the incremental assignment was really positive and the students seemed to latch on to the phase deadlines instead of seeing them as obligatory extra work.

Building an incremental assignment requires additional work from the teacher. The phases should be designed carefully to be logical parts in a reasonable and logical schedule. An 'easy start' helps the students to start working early but the following phases should not be surprisingly larger then the prior ones, as our third phase was. The students should feel the phases are for their help — as indeed they are.

7 Acknowledgments

Nokia Foundation has funded part of this work.

References

Lahtinen, E., Ala-Mutka, K. & Järvinen, H.-M. (2005), 'A study of the difficulties of novice programmers', *ITiCSE 2005, Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* pp. 14–18.

Robins, A., Rountree, J. & Rountree, N. (2003), 'Learning and teaching programming: A review and discussion', *Computer Science Education* **13**(2), 137–172.

Soloway, E. & Spohrer, J. (1989), *Studying the Novice Programmer*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Winslow, L. E. (1996), 'Programming pedagogy – a psychological overview', *SIGCSE Bulletin* **28**(3).