THOSE WHO FAIL THE INTRODUCTORY COMPUTER PROGRAMMING COURSE IN HIGHER EDUCATION

Authors: Per Lauvås & Kristine Sundt Lorentzen, Westerdals Oslo ACT

ABSTRACT

Right now, thousands of students in higher education worldwide are experiencing computer programming for the first time through an introductory programming course. After the final exam, approximately one third of these students will realize that they failed to pass the course. At the university college Westerdals Oslo ACT in Norway, we welcome 200 of these programming prospects each year. And each year, too many of them fail to pass the course. This article gives a brief overview of some of the existing research describing the phenomena before we provide our own contribution. We try to investigate the problem by interviewing students that recently participated in the course and failed. What are their stories, and what can we learn from them?

1. INTRODUCTION

Yes, we at Westerdals Oslo ACT are also facing the challenge of teaching computer programming to students who have no prior programming experience. And too many of our students fail. There is existing research describing success criteria and pitfalls when teaching the subject. We provide our own contribution by interviewing students who recently failed the course. Hopefully the students tell us something that we did not already know.

2. RESEARCH CONTEXT

Bennedsen and Caspersen (2007) studied failure rates in introductory programming courses through a web based questionnaire. 63 international institutions participated in the survey. Bennedsen and Caspersen found that 33% of the students failed. 33% constitutes a large number of failures, as the same article estimated that approximately one million students in 72 countries (not including US, India and China) enrolled in computing studies. Watson and Li (2014) followed up the research, this time “(…) by performing a systematic review of introductory programming literature, and a statistical analysis on pass rate data extracted from relevant articles.” Watson and Li supported Bennedens and Caspersens worldwide average pass rate of 67%. They also found that the pass rates had not significantly differed over time.

With hundreds of thousands of students failing the introductory programming course each year, researchers will try to understand the difficulties the students face and what faculty staff can do to improve course deliveries. Bergin and Reilly (2005) investigated 15 factors that may influence performance on a first year object-oriented programming module. They found that a student’s score on earlier subjects in mathematics and science correlated with programming performance. But the study also found that a student’s perception of their understanding of the module had an even stronger correlation with programming performance. Other studies like Boyle et al. (2002) and Ventura (2005) suggest that earlier performances in mathematical and science related subjects are not significant factors when examining student results.

We find research analyzing student results on different types of assignments (Harland et al. 2013), and how the students are evaluated. Some try to pinpoint exactly what the students find to be most challenging. Cherenkova et al. (2014) examined a dataset containing 266,852 student responses to weekly code-writing problems in a CS1 course. They found that conditionals and loops prove to be most problematic, with loops being particularly challenging.

In recent years, we find multiple articles with advices for using certain forms of teaching like inverted (or flipped) classroom. Horton et al. (2014) compared a traditional CS1 offering with an inverted offering delivered the following year to a comparable student population. Although they found that while students in the inverted offering did not report increased enjoyment and were no more likely to pass, learning as measured by final exam performance increased significantly. The conclusion was supported by further research by Horton and Craig (2015). Herala et al. (2015) describe how to create a general approach for applying the flipped classroom principles to a programming course. They suggest an approach where the students prepare for class by watching videos and reading necessary theoretical material. The students then perform quizzes to ensure that they are well prepared for the class. In class they complete practical exercises in a peer learning environment.

Others propose using student activities like Test Driven Learning (Janzen and Saiedian 2006), Code review (Hundhausen et al. 2009) or pair programming. Braught et al. (2008) reported findings indicating significant improvements in individual programming skill for students with lower SAT scores. They also found that all students are more likely to complete the introductory programming course successfully when using pair-programming. Nagappan et al. (2003) found that student pair programmers were more self-sufficient, generally performed better on projects and exams, and were more likely to complete the class with a grade of C or better than their solo counterparts. Porter et al (2013) describe pair programming, peer instruction, and media computation as three approaches to reforming CS1 that have shown positive, measurable impacts.

Some will discuss if certain programming languages should be used in the introductory programming course, or if the choice is important at all. Ivanović et al. (2015) performed statistical analysis of collected scores and grades after changing the first programming language from Modula-2 to Java. They reported results indicating that there were no statistically significant differences between the two succeeding classes with respect to success in passing the exam. This result suggested that the choice of the introductory programming language does not matter if we use students’ performance as the criterion of suitability.

Certain success criteria for teaching the introductory programming course can be investigated by using a quantitative approach. Difference in for example course deliveries or student backgrounds can be compared to exam results or dropout rates. Kinnunen and Malmi (2006) used a qualitative approach when interviewing students who had dropped out of the CS1 course. They found that there existed many cumulative reasons for students dropping out. The most frequent reasons were lack of time and motivation.

In this article, we also use a qualitative approach. We are interested in the students who did not drop out, but still failed to pass the exam. The interviews with the students are used to collect individual opinions and reflections from students having recently attended an introductory programming course, and failed. As our topic is similar to the topic of Kinnunen and Malmi, it will be natural to compare our findings with their results.

3. BACKGROUND

All students enrolled in the department of Technology at Westerdals Oslo ACT share a common first year. In the first semester they have four courses where the course “Object oriented programming 1” is the focus of attention in this paper. The students have enrolled into six different programs: “E-business”, “Programming”, “Game Programming”, “Mobile Programming”, “Intelligent Systems” and “Interactive Design”. After the initial first year, the students will go in different program-specific directions for their Bachelor in IT. The students at the programming programs will likely have a bigger interest regarding the introductory programming course than for example an E-business student.

The student results in the last years have been reasonably coherent with the findings of Bennedsen and Caspersen (2007) and Watson and Li (2014), although the average pass rate is lower. Table 1 displays the student results over the last 4 years.
<table>
<thead>
<tr>
<th></th>
<th>2012</th>
<th>2013</th>
<th>2014</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>%</td>
<td>#</td>
<td>%</td>
<td>#</td>
</tr>
<tr>
<td>Registered</td>
<td>257</td>
<td>100</td>
<td>239</td>
<td>100</td>
</tr>
<tr>
<td>Drop out</td>
<td>20</td>
<td>38</td>
<td>38</td>
<td>54</td>
</tr>
<tr>
<td>A</td>
<td>30</td>
<td>26</td>
<td>20</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>24</td>
<td>20</td>
<td>33</td>
<td>47</td>
</tr>
<tr>
<td>C</td>
<td>19</td>
<td>24</td>
<td>18</td>
<td>33</td>
</tr>
<tr>
<td>D</td>
<td>21</td>
<td>26</td>
<td>28</td>
<td>41</td>
</tr>
<tr>
<td>E</td>
<td>53</td>
<td>34</td>
<td>105</td>
<td>0</td>
</tr>
<tr>
<td>F</td>
<td>90</td>
<td>71</td>
<td>25</td>
<td>35</td>
</tr>
<tr>
<td>Drop out or F</td>
<td>110</td>
<td>42.8</td>
<td>109</td>
<td>45.6</td>
</tr>
<tr>
<td>Pass rate</td>
<td>57.2</td>
<td>54.4</td>
<td>76.5</td>
<td>60.4</td>
</tr>
</tbody>
</table>

Table 1: Student results 2012-2015.

The pass rate was below the average pass rate reported by Bennendsen et al. and Watson et al. in three of these four years. In early 2015, a decision was made to completely redesign the introductory programming course. The old course (2014 and earlier) was called “Programming 1” using Java as the programming language, but in a non-“Objects first” approach.

A new lecturer (the first author of this paper) was given the task to create a new course. In that process, existing research regarding teaching the introductory programming course was used in the planning process. Java was chosen as the programming language. BlueJ (Kölling et al. 2003) was chosen as IDE (Integrated Development Environment). The corresponding book “Objects First with Java: A Practical Introduction Using BlueJ” (Barnes and Kölling 2011) was chosen as the syllabus literature. The BlueJ community, where lecturers can provide help and guidelines, was an important factor when choosing BlueJ. As the book title reveals, “Objects first” was chosen as a pedagogical approach.

The first delivery of the new course followed a traditional approach with 2 hours of lectures followed by two hours of lab exercises each week (twelve weeks total). The students received two mandatory assignments throughout the course. The two assignments where evaluated and given a grade. These two grades accounted for 25% of the final grade (10 + 15 %). The first assignment was due for delivery 5 weeks into the course, and the second one, four weeks later. The final exam was a 3 hour written exam accounting for the final 75% of the grade. The exam was a mix of programming tasks and questions involving different educational learning goals. The students had to achieve at least an E on all three individual parts in order to pass the course. Prior to 2015, the students had three mandatory assignments accounting for a total of 30% of the final grade. And prior to 2015, the students could fail an individual part and still pass the course. As an example; the students of 2014 could fail the final exam, and still pass the course. That might well explain the high number of students receiving the grade E in 2014.

During the two weekly hours of lab exercises, the students had tutors available to provide help. The tutors were second year students who performed well on their introductory programming course (the old “Programming 1”). Choosing a new pedagogical approach (“Objects first”) and a new IDE was expected to bring some challenges. The lecturer had no experience using the IDE or an “Objects first” approach (in fact, no experience in teaching an introductory programming course at all). It was also expected to find some resilience in the tutors when using an IDE with less functionality available than what they were
used to from their own course delivery. The tutors used Eclipse (www.eclipse.org) in the 2014 delivery. When planning the new course, less functionality in the IDE was considered to be helpful for students with no programming experience as opposed to using an industry tool like Eclipse. In addition, BlueJ offers visualization of classes and objects as an important part of the objects first approach. But less functionality in the IDE was expected to be hard to sell to tutors who did not have any difficulty in using the Eclipse IDE (as they all performed well in the course). The loyalty to the BlueJ IDE was therefore addressed specifically prior to the start of the course. All tutors had to provide help using the BlueJ IDE, and no other IDE.

In order to better understand why some students had a hard time reaching the learning goals of the course, a decision was made to interview students who failed the course, or achieved an E. Kinnunen and Malmi interviewed students who dropped out. We interviewed students who were still enrolled in the study program at the university college. Hopefully, explanations from students who are still motivated to continue their studies, although they failed the introductory programming course, will provide some new perspectives to this phenomenon.

4. INTERVIEWS

The interviews were conducted by a Master’s student in Applied Computer Science (the second author of this paper). The interviewer was somewhat familiar the students after having lectured them in a course succeeding the introductory programming course. The interviewer was not involved in teaching the introductory programming course.

The interviews were voluntary, with no form of reimbursement involved. The request for interviewees was posted on an internal site for the university college. The purpose of the interview was revealed, and it was informed that the interviews would be anonymized. Only students from the introductory programming course who obtained an E or an F as the final result would be considered. In retrospect, it was not necessary to include the grade E, as table 1 displayed that no student ended up with the grade. 35 students of a total of 171 participating students did end up with an F, and were therefore possible candidates.

Originally, 16 students signed up for an interview. After contacting all the interested students, 7 students were interviewed; five women and two men. One of them revealed after the interview that she did actually get a D, but really wanted to participate. Her interview was kept as a part of the data material. The interviews where performed at campus, except one that was held through Skype. All interviews were sound recorded and later transcribed. The average length of the interviews was 12 minutes and 32 seconds, with an average deviation of 3 minutes and 13 seconds. The transcribed interviews totaled more than 12,000 words.

The interviews were performed in May of 2016, approximately five months after the introductory programming course had ended. The interviews were held as open interviews without a set list of questions to be asked. The interviewer opened with the question: “How did you experience being a student taking the introductory programming course?” All quotes are translated from Norwegian to English by the authors as the interviews were held in the interviewee’s native language. An interview guide with possible interesting topics was provided, but it was not the intention to cover all the topics. Each interview was to follow an individual path based on the stories from each individual interviewee. The interviewees were given the opportunity to add further information after the interview by contacting the interviewer, but none of them did.

The study would have benefited from a higher number of interviewees. It also would have been interesting to perform interviews on multiple universities over a longer period of time.

5. FINDINGS

As expected when interviewing 7 students, we got 7 different stories of what went wrong when they tried to learn how to write computer programs. Some explanations vary, some can be found contradictive and
some share common traits. It is expected because existing research describes a multitude of different elements affecting student results in such a course. When quoting students we have numbered them S1 to S7.

5.1 Contradictive stories on syntax

Two students described what can be viewed as contradictive reasons for failing the course. The stories involve syntax.

S2, on what was most challenging: “It's finding out what all the different syntaxes are, you have to figure out how to write the different things, somehow, because, every time you write something you have to figure out what it is. It's not like that for example: if you do math, so you can divide, and on the next task, I can also divide. I felt it was not like that, it was like, you have to learn something new every time you learn.”

S5 talking about syntax: “(...) the hardest thing for me was like, to know when I should use a “for loop”, and such things. A little more, well, what should I say… I manage somehow… I manage to write it, but I do not quite know why it is correct. I cannot think programming, which is why I failed.”

5.2 Terminology and multiple languages

Two of the students described challenges with a large amount of new terms being used in combination with an English book and Norwegian lectures:

S6: “It really ended up with me becoming more and more confused by these abbreviations, and the fact that the book was in English. The teacher spoke Norwegian, the lectures were in Norwegian, and the book again… When I would read something, it was in English, so it was like, "Okay, translate this please!" (...) When the book is in English… To try and translate it and try to remember what the teacher also said and see if "Is this the word in the book?" I wish that the book was in Norwegian, or that the lectures were in English, possibly with a recap in Norwegian, with Norwegian-English translations, but it might also become difficult for some people.”

S4: “And so the books are in English, and I think it is very demolishing, because there are lots of words we have not learned and we know only basic English, we do not know that damn...”

5.3 IDE

As expected, not all students were happy with our use of a new (compared to the previous year) IDE: BlueJ. We especially predicted that the second year students, who would work as tutors in the class, would find it somewhat difficult to tutor the students in an unfamiliar IDE. However, only one of the students described challenges with using the chosen IDE:

S4: “The subject itself is not that difficult, it was more that I struggled with BlueJ, which made me sort of just, yeah, "Fuck it all", somehow. Because I worked very much on Treehouse, so I know pretty much but it does not help to somehow learn things on Treehouse and come and do things in BlueJ, for my part. (...) So I lost somehow the motivation when I did not understand the program, so I think switching to something that is not BlueJ would have been much better. (...) I think it would be much easier if we had gone for the same java style they used before. Then it's like… I've talked to a lot of the tutors and there are many of them who do not really understand BlueJ.”

Treehouse (teamtreehouse.com) offers a multitude of online courses, and some of these are computer programming specific.

5.4 Learning environment

The students described a good student environment at the school in general, but those who lagged behind in the course curriculum found it hard to seek help from tutors, the lecturer and fellow students. Some of them also describe an awareness of other students knowing more than them.

S1: “So for me, the first exercises in the first week were also a great challenge - to understand them. I did not understand any of it. While for others, who had seen these things before, it was so easy, right?”
S1 on help from tutors: “(...) when I first began to lag behind, then there was sort of like this... The lab exercises... In Chapter 5, for example, when, when ... When something is unclear from Chapter 2, then it's sort of embarrassing asking too, right? You would not want to go back there like that, like: "Hah, you do not even know that?" Right? So I was like... I did not want to attend there anymore.”

S1 on working individually or in a group: “Individually, because it's like, when you know nothing, then you sit down with someone else who already knows something, right? So they will work from where they already are, and not return to the first lecture and work from there. So then you become sort of a loner, right? It's just like that. But if I had known more, I would have had more confidence to sit with those who know more, and work with them and learn new things. But when you know nothing, then it becomes somehow: "Hah, you do not even know that?" Right? So then you become... You isolate yourself naturally, so to speak.”

S1 on asking questions in class: “Someone asks questions, yes, but those who ask are the ones that know a lot initially, who are just wondering about something that I do not even understand, right ... They ask, they ask about everything, so ... They are confident in the discipline, right? So then it is much easier to ask, because you understand the response from the teacher too, right? So, those who know everything... If you then ask the lecturer to repeat something over again, it's like "Ugh!" for those who already know it. They do not understand why others do not understand it because you understand it, right? (...) Although the environment at the school is good: It's awesome, oh yes, student life, and the class, and we've never had any conflict or anything like drama, not at all. So, in that respect it goes really well.”

S3 on working in a group: “Yes, I worked mostly in groups, but they, well, the people in the group, they were like, they got it, and I was like... Then I cannot ask them now. (...) So, yeah, so it was like they understood things right away and were very like "Yes, I got it, I got it!". And so it was really like: "Do not tell me the answer, do not tell me the answer!" and I just... If they say "Do not tell me the answer, I will find it myself", then I come and just ask... So I was kind of like "Ooohh". It was like, the group understood a lot, and so I was like "Whaaat? What are you doing? What happens? What do I do?"

S4 on different level of knowledge within the student group: “When you are not any good, they're like "Oh my god, you are so stupid!" and it's like *sigh* "Sorry that I did not sit and spend my time at home coding. I have other interests.""

S6 on asking the lecturer: “(...) if I do not want to go up to the teacher and ask "I'm struggling with it, this and that," perhaps, for my part, rather be using an online forum in our LMS and used it a little more actively. (...) Sometimes I wish that we could just: "Hey, all you who are afraid to say anything please raise your hand. Come, we shall have a joint meeting, and we will talk about why we are afraid of talking in public.”

5.5 Falling off

A common theme from the majority of the interviewees was stories describing how they came to a point where they no longer managed to understand what was being lectured. When that point was reached, it was very hard (or impossible) to catch up. Some of them described the point as in the beginning of the course. Others managed to keep up with what was being taught for a while, but then later on, as some of them describe: “fell off”.

S1: “It is like... When I first started lagging behind, I was behind, right? So all the things I learned after that, I did not get any of it. So then it just became a bit like dominoes. (...) so how could I pick up from Chapter 4 when I did not understand Chapter 1?”

S2: “For my part, it was really quite okay in the beginning. Then I started falling off and when you... When I first fell off, it was so difficult to get into it again. No matter if I felt that the lecturer did well, because I had not gotten to the point yet, and then it was very easy to end up behind.”

S3 on when it started to become difficult: “Maybe the third lecture. The first one went quite smoothly, and the second one too. But then on the third I started to just, I was a bit like... I could not really manage to pay attention anymore.”
S5: So, yeah, I kept up for a good while, and then I started lagging behind. And so, it was really like I was just a little too late to catch up again. (…) I started to lag behind, and then it was so easy to just quickly give up when I tried to understand the curriculum for the next lecture. So, it was really like that: it just accumulated. (…) Finally it was like I just gave up. Because I just felt like… I could not manage this. Like, shit…” (…) eventually it became somehow that it went in one ear and out the other. I forgot about it as soon as I had done something, and then it's like… It's easy to give up, anyhow ... Feel sorry for myself and all that. (…) But, yes, as one begins to fall off, it's like… It just builds up, and then just everything becomes a little less motivating.

S6: “If you first fell out, it was not easy trying to get in again. That was really what was the most difficult.”

S6 on when it started to get difficult: “After submission 1. I ended up with a very powerful flu, so I was not there on the actual delivery and it was sort of like… Just in those two weeks it was like... Up until submission 1 and after submission 1 it was like... "No."”

5.6 Reasons for «falling off»

Multiple reasons for «falling off» were described, and we have seen some of them already. It could be difficulties using an IDE. It could be certain elements in the learning environment, it could be trouble understanding the many new words being introduced (in multiple languages) and any combination of these. But one other reason was described by some of the students. The two mandatory assignments accounting for a total of 25% of the total grade could also affect the student’s ability to catch up with the lectures:

S1: “And then there were the mandatory assignments. We received grades, right? So, then it was like… You got stressed out because you wanted to get the best possible grade so that it would not deteriorate your final grade. So then it was like, do whatever to get the delivery done and get a good grade. If only one could get the good grade, as we knew it would affect the final grade, right? So then, the focus was not on solving the problem and learn something, and like, possibly fail and then learn from the mistake. It was focus on getting the best possible grade, if not it could affect ... At the end. (…) Yes, you focused on the specific problem, and not how to solve that kind of problem in general or how the computer programming worked in the assignment and what we could learn from it and... It was just… I have to get an A, or else it can turn out bad if I get a D or E or something, right?”

S3 on asking tutors for help: “It might have something to do with the mandatory assignments. When doing the deliveries, then I asked more for help. But then I sort of got the impression that, you know… They could not help me too much, so then I was a little like… Then it was like… I felt that, there is no point in asking questions. However, it's not like that in regard to the other exercises, but I think I was a bit freaked out because of it. But I understand that they cannot help us like that… They cannot provide the entire solution. It's simply not possible, because then there is no reason to have the assignment, but yes… So no, I do not think I asked for help that much. Just a little bit, maybe.”

S6 on delivering assignment 1: “Well, I delivered it. I got C or B on it, I think. It was something like that, so I was very pleased with that. But it was like… When we were to move on, then it was like ... No... I sort of got so much help from my classmates that I would not have been able to solve it myself. And when I then suddenly sat down and was to do something like that myself, it did not work out.”

So, we see that the focus on good grades on the mandatory assignments will result in some students seeking help from other students. It can also result in some levels of copying:

S4: “So, in the first semester, I did all the deliveries myself. In the second semester, I have just gotten them. I have just acquired an A delivery and messed it up to a C delivery, right? I see the logic. When I acquire an A delivery, then: "Oh shit, this I understand!", but I cannot submit it. So, to not cheat, I get it a bit down to my level.”
"
6. DISCUSSION AND IMPLICATIONS

We should be careful not to generalize after reading stories from 7 students who failed to achieve a decent result on an introductory programming course. But the stories will tell us something about what went wrong for these seven individuals. Some of the causes for their individual results can be challenged, and some are more difficult. For example, stories describing illness or students children’s illness will be external events that are less interesting to focus on.

6.1 Syntax and terminology

Challenges involving syntax and introduction of new terminology (in this case in multiple languages) can easily be addressed. More focus can, for example, be put on creating and maintain a course relevant syllabus. In case of lectures being held in another language than other important course recourses (in this case the book on the curriculum), emphasis can be made to use a stringent vocabulary so that the students will experience less confusion. But as terminology in a programming language is not chosen randomly - they are linked to our natural language - translations are important. For example: What does it mean to assign something in Java? What does the term “assign”, really mean? If we translate it, what is our translation of it? Will it help us to understand what it means in our natural language when faced with a computer programming context? Probably a translation to the native language of the students will be beneficial.

6.2 IDE

The choice of IDE to use in an introductory computer programming course can be discussed. When an “Objects First” approach with Java as the programming language was chosen at our school, the choice of IDE was fairly easy. BlueJ is an IDE with much less functionality than an industry IDE like Eclipse, IntelliJ or NetBeans. Eclipse, which was in use the year before, provides much more functionality. It was therefore believed to be harder to start using an industry IDE, especially for those with no prior programming experience. BlueJ is easier to start using, and provides good visualization of objects. Instead of starting lecture 1 with trying to explain the Java main method signature, the students start using already written programs and can visually see the objects in the program.

It is therefore reason to believe that a student failing an introductory course would NOT have benefited from using another IDE. When transitioning from Eclipse to BlueJ in a course delivery, one must suspect that the second year students tutoring first year students in an, for them, unknown IDE might find it challenging. This was, as we have seen, also supported by the interviews. But that will only be a challenge in the first year after the transition.

6.3 Learning environment

Actions can and should be made to create a good learning environment in a computer programming educational setting. Some students are reluctant to raise their hands in class or ask a tutor or classmate for help. Those with little experience in the field easily detects that other students know more than themselves, and it makes it harder for some to ask questions as they feel it would make them look foolish or dumb. These findings are supported by the works of Barker and Garvin-Doxas (2004 and 2005) who described a competitive and defensive climate in computer science classrooms. We consider multiple adjustments to meet this challenge.

In large lectures, we may use an application where students can ask questions anonymously. This can be beneficial as a student will no longer be needed to speak publicly in a large class of students. Another benefit is that the lecturer can filter non-appropriate questions. Questions that are outside of scope will be dismissed. The same applies to questions of such difficulty that only a few of the students will find any value in an answer. Hopefully, the use of the application can result in more struggling students asking questions in lectures. It can also result in a higher self-esteem for the students who no longer will hear overly complicated questions being asked in class. Both questions and answers can be stored so that students and faculty staff can revisit them later.
During lab exercises, we might introduce specific labs (rooms) dedicated to those students who are struggling the most. The lab can have a higher density of tutors, and more emphasis can be put on creating an atmosphere where all students will be able to ask questions without feeling "stupid".

6.4 Evaluation

Our motivation for initially introducing summative evaluation of the two mandatory deliveries was based on the idea that the students have to be somewhat forced to start programming. Computer programming is something that cannot be understood by simply reading about it. All students must start coding in order to learn how to write code. By using summative evaluation it was believed that the students would work hard on the given assignments. The possible side-effects described by these interviews were not considered.

It is far from ideal that the mandatory deliveries steal too much attention. Students might not be able to perform other parallel activities necessary to keep up with the course (or other parallel courses, for that sake). The deliveries have to enhance learning. For some of our students, they are clearly not. For some of our students, it increases the change for them to, as they say: “fall off” the course. And when they are off, it is very hard to get back on. This can be supported by Robins (2010) who describes the Learning Edge Momentum (LEM) effect in a CS1 context. Students who have no prior experience in computer programming and initially achieve successful learning will find it easier to continue learning. Unsuccessful learning makes new learning harder.

After a completed course, the students end up with a grade. The grade should reflect how the student performed according to the learning goals described for the course. As the grade reflects the end result after completing a course, using summative evaluation can be hard to justify. Why should a student get a part of that grade when the course is not yet finished?

In order to address this, we have decided to keep the mandatory assignments, but will no longer provide a summative evaluation. The students will receive formative evaluation on the two mandatory assignments. As the assignments will remain mandatory, the students will have to deliver them in order to gain admission to the final exam.

When holding on to mandatory assignments, and require two accepted assignments in order to obtain access to the exam, wouldn’t the same challenge remain: Students being overly focused on the assignments in order to not fail the course? It depends on what we require for a delivery to be accepted. In the next delivery of the course we plan to use the same type of assignments as last year. Some computer program has to be written or modified, and provides some sort of functionality. But the student will not have to complete the program. If the student does not manage to complete the assignment, the student must provide an explanation of what is preventing the student from completing the task at hand. Is there a problem using the IDE? Is it a problem with syntax? Is it a problem understanding the assignment? The student can also be asked to describe how the student has been working on the assignment. Has he worked alone or in a group? Has he been using material from the lectures? Has he read the relevant chapters in the book on the curriculum? By providing this information, the student will have the assignment accepted. The student will also get important guidance on how to work in order achieve the desired learning goals set for the course.

The explanations will give the lecturer important feedback on what some students might be struggling to achieve. As there will be no grading of the delivery, there can be full openness about possible solutions. These can be discussed in class along with common difficulties described by the students. By doing so, we believe we can get the students started with writing code, and enhance learning also for those who are really struggling.

6.5 Kinnunen and Malmi revisited

Kinnunen and Malmi used a questionnaire and interviews to gather information on reasons for students to drop out of the CS1 course. 105 of the 212 students who dropped out answered the questionnaire, and 18 students were interviewed. The interviews suggested that there were two major reasons for dropping the course: 1: No time. 2: No motivation. The lack of time was further categorized in three subcategories: 1.1: The student decided to prefer doing something else. 1.2: The student had not booked enough time for the
course in the first place. 1.3: Some parts of the course were more difficult than students expected and therefore the course took more time. “No motivation” was also subcategorized: 2.1: No study motivation in general. 2.2: Payoff is imbalanced and therefore motivation drops. 2.3: Some parts of the course were too difficult and therefore motivation drops.

The results from Kinnunen and Malmi differ from the results in this study. The only explicitly expressed example we found regarding lack of time or motivation was S4 stating that she lost her motivation because of the BlueJ IDE. Although lack of time and motivation is not explicitly described as a reason for failing in our study, it can be still be important factors. Maybe students in our study describing how they “fell off” and could not get back on really struggled with putting in enough hours to keep up? And maybe lack of motivation could explain some of the problems our students described, although motivation was only mentioned once?

But instead of speculating if underlying factors are in play or not, we focus on the data material provided by our own students. It should be reasonable to believe that there will be different results when Kinnunen and Malmi interviewed students who dropped out, while we interviewed students who participated in the entire course, but failed to pass the exam. Differences are also expected when the two courses have differences in content and delivery, although the overall topic is the same.

7. CONCLUSION

After interviewing students who failed to pass the introductory computer programming course, we identified several reasons for students to fail. Some of these findings supported existing research describing the same phenomena. As the students described their experience in a specific course delivery, the findings are especially relevant for a similar learning environment. When using mandatory assignments in addition to a final exam, it is our recommendation to not use summative evaluation. It may cause students to focus too much on the task at hand. The strong focus may steal too much attention from other important parallel activities. That loss of attention can cause the students to longer be able to keep up with the course curriculum. And when the student is lagging behind, they find it very hard to catch up.

We propose a way of keeping mandatory assignments and also enhance learning. When evaluating a mandatory assignment, we can accept it although the evaluation criteria are not met. But only if the student who fails to complete the assignment delivers an explanation of why the attempt failed and what type of help will be need for him to complete it. By doing so the assignment becomes a work requirement, not a completion requirement. The lecturer will receive important feedback on what some students are struggling with. And why should an assignment delivered midway through a course account for part of a final grade anyway? The final grade should reflect the final learning outcome, and not how quickly the student managed to learn it.

It is important to recognize that introducing these changes can lead to unwanted side effects. Maybe using summative evaluation actually helped some of our students to actually get started coding? Maybe some students in the next course delivery will take the easy road by simply describing their problems to get the work requirement accepted? We will therefore follow up this article with a new paper evaluating the changes.

8. REFERENCES


