Assessing Undergraduate Computer Programming Skills

Ewan Klein

November 4, 2013

Contents

1 Introduction

During the academic year 2012/13, I was seconded to the Institute for Academic Development[1] from my home department, the School of Informatics[2]. The main thrust of the secondment was to examine issues around teaching and assessing computer programming skills, with a focus on first year undergraduates in Informatics. However, during the course of the secondment, in part due to discussions with Velda McCune and Jon Turner, I also invested time into a couple of other initiatives—InfoPALS (a peer-assisted learning scheme) and the Smart Data Hack[3] —that were complementary to my original goals and which had not been part of my initial plan. However, in this document, I will focus on one component of my work on assessing computer programming.

2 Teaching and Assessing Programming

There is a considerable body of work on the pedagogy of computer programming, and recently a lot of attention has been paid to the notion of computational thinking[4] as a key competence; see for example these slides by Quintin Cutts[5]. In this context, computational thinking involves two aspects: analysing a problem at different levels of abstraction; and breaking down its solutions into steps which can be defined in an explicit, 'mechanical' way. However, programming involves more than this. An acquaintance with basic maths is helpful (and is required of all students in the School of Informatics), together with some degree of problem-solving ability. In

[1] http://www.ed.ac.uk/schools-departments/institute-academic-development
[2] http://www.ed.ac.uk/schools-departments/informatics
[3] http://www.inf.ed.ac.uk/student-services/committees/teaching-committee/meetings/6th-march-2013/ILWHackReport.pdf
[4] http://www.cs.cmu.edu/~CompThink/
[5] http://www.slideshare.net/compatsch/quintin-cutts

addition, learning to use a programming language has some similarities with learning any foreign language, say Italian or Chinese: you need to acquire the basic vocabulary, some syntax and some idioms. And you need to practise, practise, practise!

My main experience of teaching programming has been acquired, sometimes painfully, through running a first year programming course Object-Oriented Programming (INF1-OP)[6] Informatics—in effect, an introduction to Java programming, with an intake of between 150 and 250—at the School of Informatics for 4 years. This was a baptism of fire for me: it was the first time I had been responsible for a large UG1 course, plus I was also a novice at Java: I had to learn enough to be able to teach it. After a number of experiments and tweaks, the structure of the course ended up as follows:

- one 1-hour lecture per week
- one 2-hour lab slot per week
- one 1-hour tutorial per week for a semester-long team project
- a 3-hour end-of-semester practical exam

The lab slot focussed on students solving a graduated set of programming exercises in the Informatics computing labs, and feedback was provided in two ways: a team of demonstrators was available to answer questions and to proactively engage with students who seemed to be stuck; and solutions to the exercises were automatically marked for correctness using an in-house system called Infandango. In general, my bias in teaching programming was towards learning by doing, and this fitted in well with the fact that the final exams themselves required students to write code.

3 Infandango

As mentioned above, the Infandango code automarker was developed within the School of Informatics by the two Teaching Assistants on the INF-OP course: Mike Hull[7] and Daniel Powell[8]. As well as marking code submitted by students, Infandango also functioned as a web-based platform for publishing exercises.[9]

Figure 1 shows one of the overview pages in Infandango. In order to interact with Infandango, you have to be logged in, and what you see will be adjusted to your history of working with the system. In Figure 1, the lefthand pane presents an expanded view of all the exercises available during week 2 of the course, while the centre pane introduces the exercises, and shows which ones have been successfully completed so far by the user.

[6] http://tinyurl.com/inf1-oop

[7] http://www.anc.ed.ac.uk/dtc/index.php?option=com_people&func=showall&userid=359

[8] http://homepages.inf.ed.ac.uk/s0347677/

[9] For a brief overview of Infandango's design and architecture, see http://hdl.handle.net/1842/5206. The code itself is available as an open source project from https://bitbucket.org/ewan/infandango. Installation and basic administration documentation can be found at http://infandango.readthedocs.org.

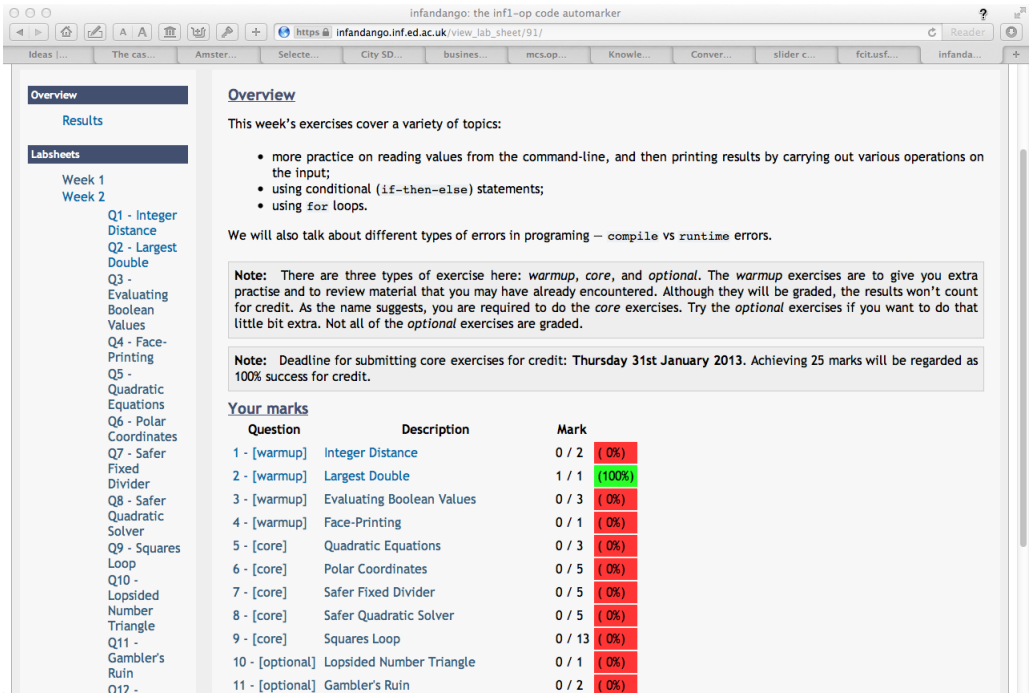


Figure 1: Infandango Overview Page

Let's see what might have been involved in answering Q2 (*Largest Double*). The page for this exercise is shown in Figure 2. At the bottom of this page, you can see that students are asked to write some Java code that meets the stated requirements, and then to upload their answer via the web. Once this has been done, the student gets taken to the page shown in Figure 3. In this case, the submitted answer is incorrect, and scores 0 marks, as shown in Figure 4. By clicking on *Result of Test*, they should be able to receive a more detailed analysis of why their code failed to work properly.[10] After a period of embargo (typically about one week after the exercises are released), they should also be able to access a model answer. Students are allowed to resubmit their candidate answers as many times as they wish, and Figure 5 shows the page when the third submission is correct.

4 Infandango Analytics

All the information about exercises that Infandango presents on the webpages is stored in a database, and likewise all the information about the tests that are run and about code that students submit is stored. In 20011/12, there were eight weeks of coursework exercises, each of which comprised around five distinct exercises; each exercise typically involved running several tests against the submitted code; students could submit multiple times for the same exercise; and practically all of the 120 students enrolled submitted at least one exercise. Towards the end of the 2011/12 semester, we noted that Infandango had executed 58,000 code tests. Consequently, it gives us a rich resource of data to study. During my secondment, I worked with two Informatics UG4 students who were carrying out projects on the (anonymised) data collected during academic year 2011/12,

[10] The feedback that a student receives is based on the results of running tests against their code. By default, the level of detail returned if the test fails is relatively rudimentary, and the onus is on the writer of the test (e.g., the course instructor) to write more informative feedback.

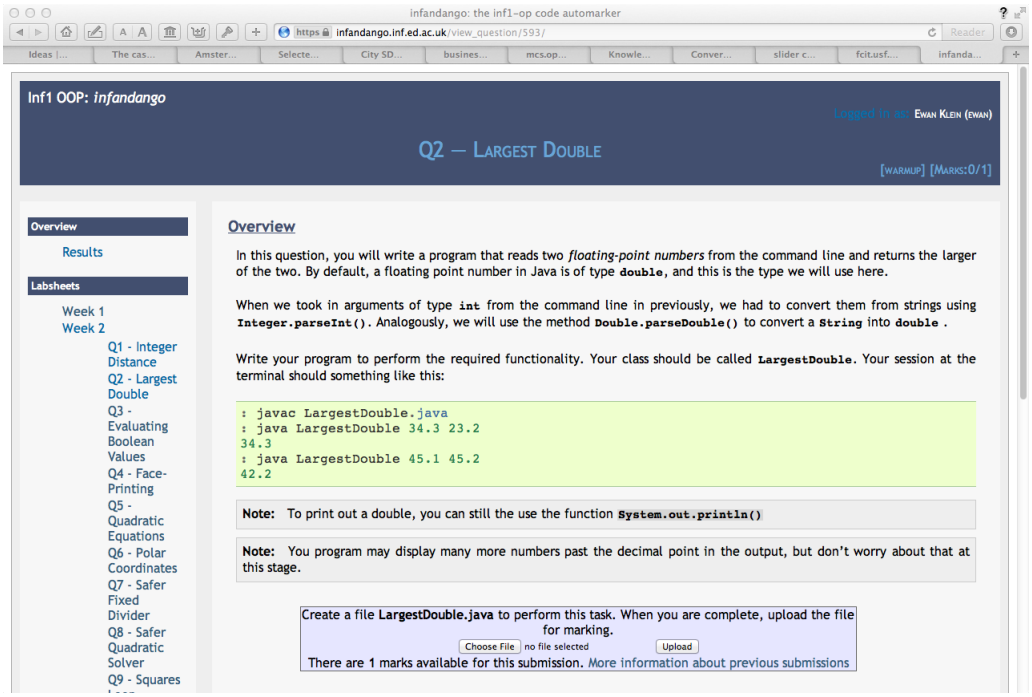


Figure 2: Infandango Q2 Page

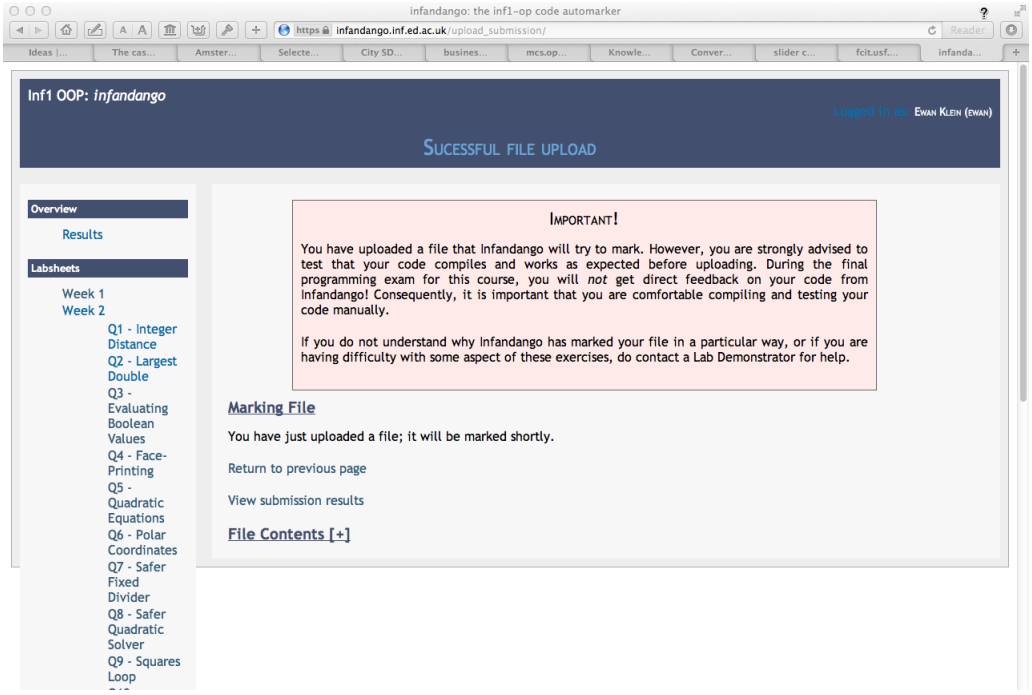


Figure 3: Infandango Submitted Page

namely Joe Bazalgette and Paul Thomson. One of the projects (Bazalgette) interested in exploring a range of

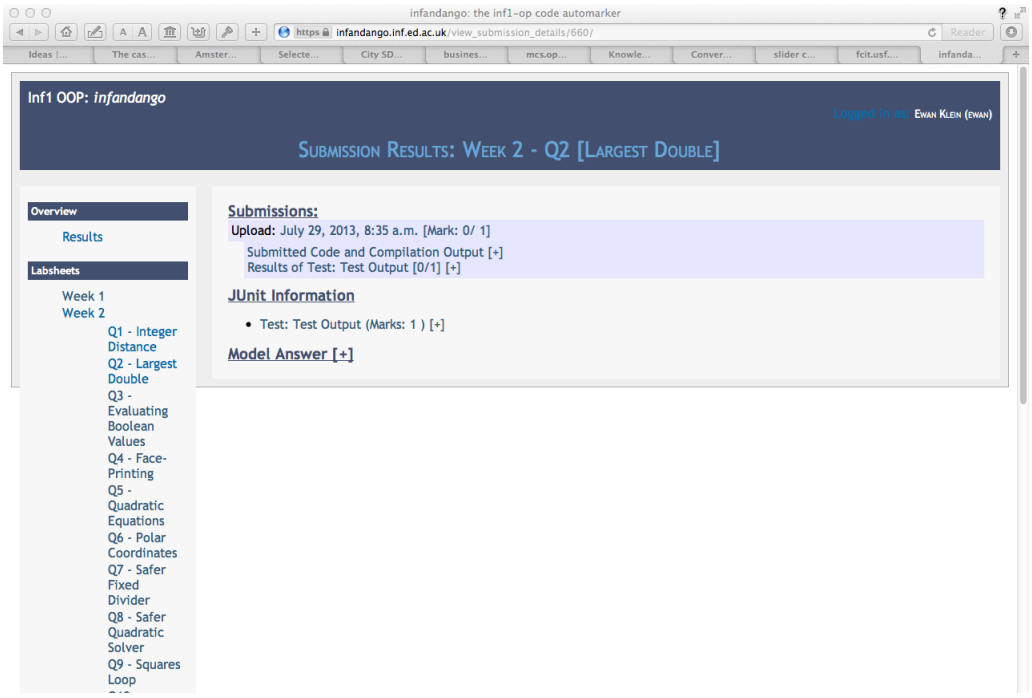


Figure 4: Infandango Feedback Page

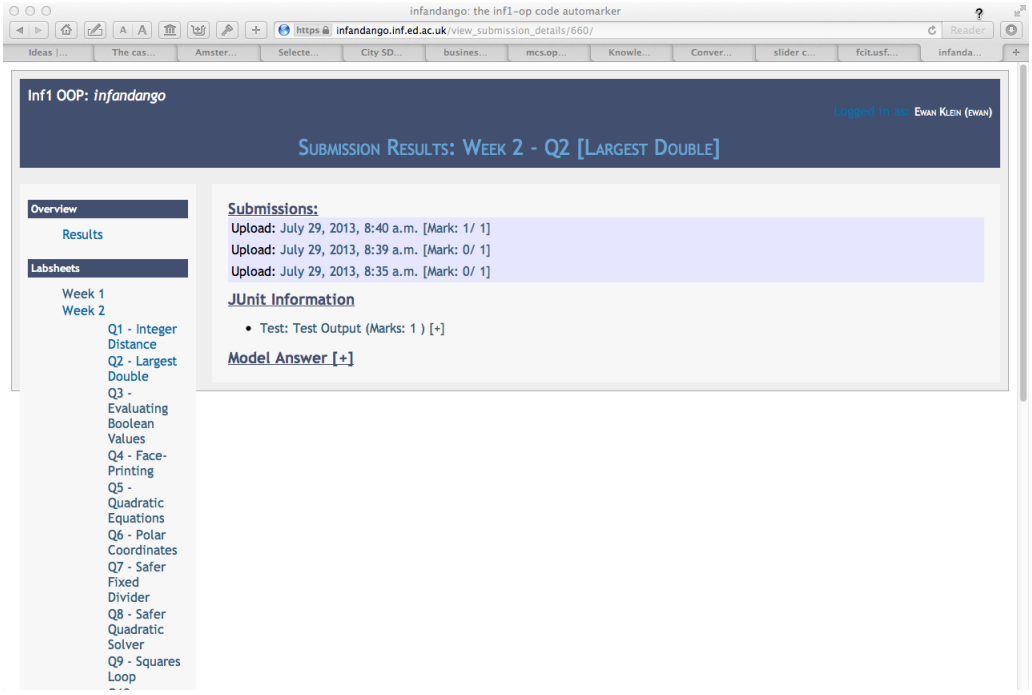


Figure 5: Infandango Feedback Page

correlations in the data, such as between the number of submissions per question and exam success, and also

asking whether it is possible to discern significant sub-clusters of students based on their mark profile. The second project (Thomson) employed machine learning techniques to predict how well a student was likely to perform on a given question given their performance on the preceding questions; this was intended to form the basis for giving students accurate feedback on their progress. In this report, I will cherrypick some the more striking results from Bazalgette's project report (*Exploring Infandango: Applying Data Mining to an Automatic Java Marking System*).

One of the first observations was that there was a strong correlation between number of submissions and the overall marks that students received for their coursework exams; that is, more submissions meant higher marks. This of course seemed quite gratifying. Less pleasing was the lack of correlation between coursework marks on Infandango and the marks that were independently collected for the end of course exams, as shown in Figure 6.[11] We were curious to see whether there might be stronger correlation if we restricted attention to

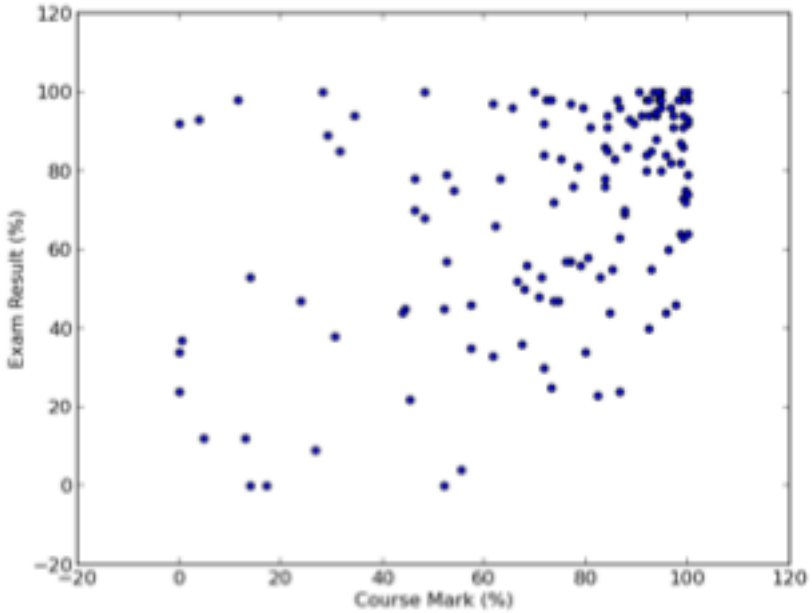


Figure 6: Coursework marks fail to correlate with exam marks

students without prior programming experience, however this also failed to hold — see Figure 7.

[11] I apologise for the poor quality of these graphs, which were cut-and-pasted from Bazalgette's final report.

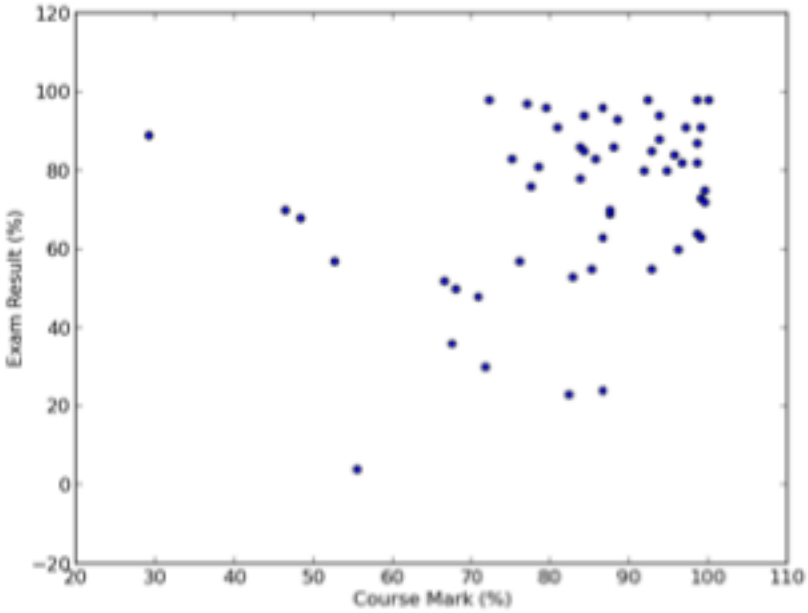
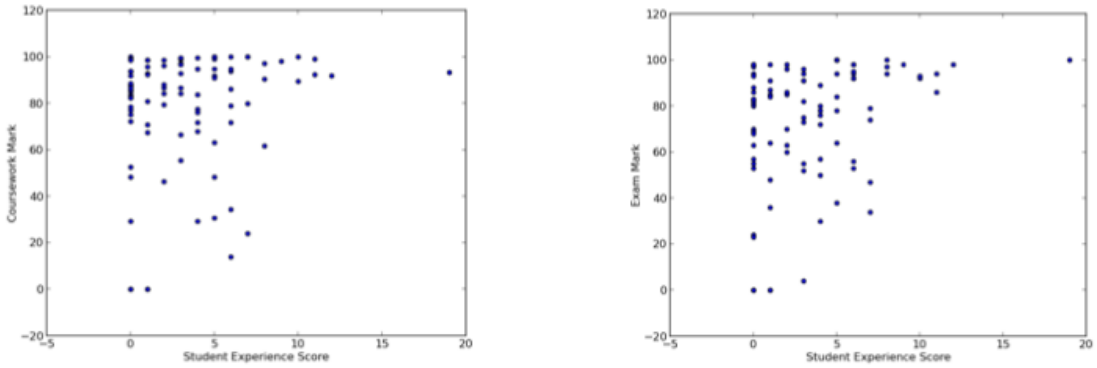


Figure 7: Coursework marks fail to correlate with exam marks for students without prior programming experience

As a short digression, I had also considered the hypothesis that exam performance would correlate strongly with the level of prior programming experience. In the worst case, this might mean that the performance of students with little experience was not greatly assisted by the course, while experienced students might have achieved good results quite independently of the course. Although experienced students did indeed tend to score well in both the coursework and the exam, performance levels by students with little or no programming background were distributed relatively well across the group. That is, Figure 8b shows that a fair number of students, i.e, represents by points around the origin of the x axis, scoring 80% or above.



(a) Coursework (b) Exam

Figure 8: Distribution of results vs. different levels of programming experience

One concern which surfaced periodically during the labs was that a minority of students seemed to be minimally 'fixing up' their code to avoid the errors that Infandango was flagging, rather than thinking through the deeper issue of designing their code to address the requirements of the exercise. Students who were caught in a loop of successively tweaking superficial aspects of their code in order to satisfy Infandango might be expected to submit more candidate answers to a given exercise than those who analysed the task at a more conceptual level. Some evidence that supports this hypothesis is shown in Figure 9. This shows an interesting cluster of

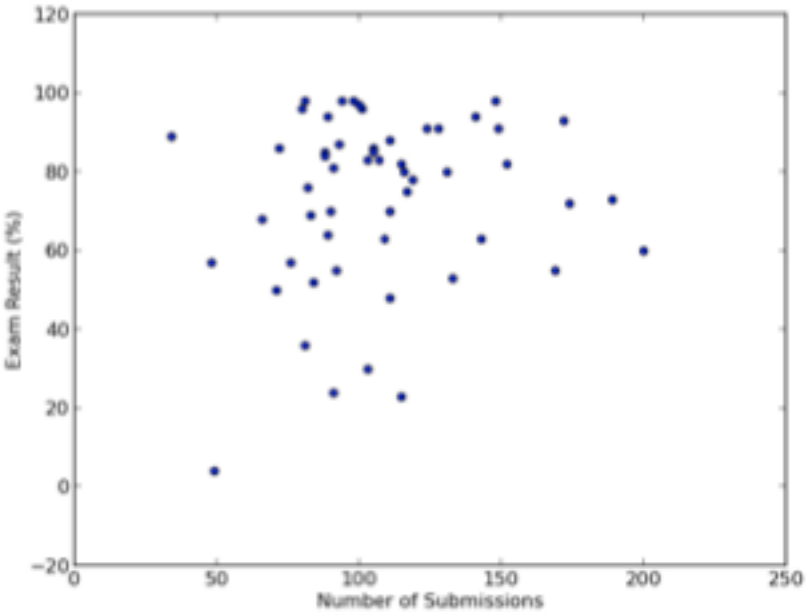


Figure 9: Higher submission rates do not always equate to exam success

around 25 students who made around 75–125 submissions and achieved around 75% or more in the exam. By contrast, about half of the students who made more than 150 submissions achieved less than 75%. This may be reading too much into what could be chance distribution of results. Nevertheless, it would be interesting to probe this effect more deeply, in particular by properly identifying which of the submissions were *resubmissions* to the same exercise.

It's worth noting that modifications could be made to Infandango to discourage students from getting stuck in an ineffective learning strategy, such as adding allowing the system to block further resubmissions for a fixed period once a threshold had been reached.

5 Conclusion and Future Steps

Overall, Infandango proved to be a powerful and effective framework for publishing lab exercises, marking student submissions, and recording the results for reporting and further analysis. The design of the system is modular, and is not restricted to a particular programming language. For example, another UG4 project by James Vaughan successfully build a plug-in that allows Infandango to process student code written in Python rather than Java. Nevertheless, there is a range of improvements that could be made to the system. These include:

- Making notification of coding errors more informative for students.
- Presenting students with different problems, depending on the outcome of tests.
- Using analysis of student submissions to trigger support and remedial intervention by the lab demonstrator, tutor or course instructor, as appropriate.

One of the original goals of the project carried out by Bazalgette was to use the pattern of errors in a student's submissions to identify underlying mistakes in their conceptual model of the programming task. For example, it should be possible, in principle, to recognise that someone is struggling with exercises that require certain kinds of 'loop' constructs in the program. Although we didn't achieve this goal, it still strikes me attainable and worth pursuing in the future.