PTAS Project Report (for REGULAR PROJECT GRANTS)

Project Title:

Project type : Research Project
Principal Investigator : Paul Anderson
Schools/department : Informatics

Team members (including Schools and Departments) :
Paul Anderson, Ross McKenzie, Cristina Alexandru (Informatics), Anna Wood (Education)

For further details, please contact: Paul Anderson <dcspaul@ed.ac.uk>

Project teams must submit a report within 4 months of the conclusion of their project.
Copies of dissemination material (eg journals/newsletter articles, conference papers, posters should be listed and attached (separate to the word count). The brief report will be published on the IAD web pages.

Report (maximum 1500 words)

What did you do?

We created an implementation of an "Adaptive Comparative Judgement" algorithm, together with a web interface for the Informatics MSc course on Java programming. We used this to allow students to view and compare other students' programming assignments (anonymously) in terms of code readability. As well as making pairwise comparisons between the code samples, we encouraged the reviewing students to leave free-text comments, which were then passed back to the authors.

We used the ACJ algorithm to derive a ranking for the submissions, based on the students' pairwise comparisons. We also used demonstrators to double-mark the assignments for readability, so that we could compare the ACJ rankings with the explicit markings.

We then interviewed a number of students and demonstrators to (a) explore what the students understood by the term "readability" as applied to code, and (b) to determine if and how the comparison exercise had helped in their understanding of code readability. These interviews were structured around a “think aloud" exercise where the participants were asked to talk about their thoughts as they read through one of the code samples. This was followed by some general questions about their impressions of code readability. We also provided a feedback questionnaire so all students could comment on their experience.

What did you find out?

Despite the success of ACJ algorithms reported in the literature for similar situations, this failed to produce a meaningful ranking for our assignment. The manual double-marking did produce a reasonable correlation, however there was not a large spread between the marks and most submissions fell within a small marking range (although the reasons were often different). We conducted some simulations with the algorithm which seemed to suggest that it would not produce a good ranking under such circumstances. However, we are continuing to explore the reasons for this, and possible improvements to the algorithm, and the overall process (for example - the size and choice of the samples presented to the students).

The interviews, and written feedback produced some interesting results. In particular:

1. Students found the ACJ exercise useful, but having to make 10 comparisons (which was necessary for the ACJ algorithm) was too much. The resulting fatigue may have contributed to the poor ACJ ranking.
2. Students did learn a variety of useful things from looking at other's code. This included readability issues such as naming conventions and better ways to structure code, but it also included additional aspects such as the actual algorithms being used.
3. Students benefitted from seeing very good code and very poor code (particularly during the think aloud exercise) but not from comparing two pieces of code that were very similar.
4. Some students found giving feedback helpful but were not clear on the criteria that they should use for these comments.

There were some technical issues which mis-attributed some of the student-provided comments, making it difficult to reliably judge the effectiveness of this part of the exercise. However, some students did report that they received useful feedback, although we felt that much of this was rather too general to be really useful, and the students may have benefitted from some guidance on giving constructive feedback.

In conclusion: there are clear benefits from students being required to read and understand other students code - both "good" and "bad". However, these benefits are best realised when the code samples being compared are (a) not too similar, and (b) studied in some depth. These requirements conflict somewhat with the ideal requirements for the ACJ ranking algorithm which tends to present similar samples and only requires an “A" or "B" answer.

How did you disseminate your findings?

The ACJ software has been made available in the public domain: https://github.com/RossMcKenzie/ACJ.

Before publishing or presenting further results, we would like to explore further the reasons why the ACJ algorithm did not yield a good ranking - we intend to use the same process for next year's course (perhaps in a different context), and we also encouraging others to run similar exercises.

What have been the benefits to student learning?

Especially in computer programming, but also in many other fields, students learn a great deal from seeing real examples (both good and bad) of other's work. This is difficult to arrange in a course where most of the student work is assessed and cannot be shared. The process that we used incentivises students to study other's work, after that work has been assessed.

Although the ACJ algorithm failed to produce a ranking - which we had hoped would support assessment - this failure may be indicative of underlying issues, such as a lack of clarity on the assessment criteria. Understanding this better will help us to improve the assessment process.

How could these benefits be extended to other parts of the university?

The core ACJ software has been made freely available for others to use. We are currently developing (as part of an MSc project) an interface to this software which should make it easier for others to adopt and use in a different context.