A Novel Use of Individual Code Reviews to Improve Solo Programming on an Introductory Programming Course (1000 Words)

Dr. Glenn L Jenkins

Programming is a difficult subject for many students and therefore a popular topic in computing education research, with extensive research into the teaching and learning of programming [1]. Collaborative learning approaches are desirable as they fit well with the needs of industry [2]. Such approaches have recently been used [1-3] to combat high attrition rates (common on programming courses [2, 4],) along with pair programming [5-7] and other techniques [8].

Peer code reviews (or code inspections,) have been successfully applied to the teaching of programming [2, 4, 9]. Code reviews can also be applied in a individual context as in the Personal Software Process (PSP) [10]. Students at present undertake pair programming (which has an element of code review [2],) and work on a number of collaborative exercises on this basis. The utilisation of pair programming raises a number of concerns, firstly pair breakups and their effect on retention [11], secondly difficulties associated with identifying the contribution of individual students to joint assignments [6] and finally the effect on progression (some student may struggle to make the transition between a first year of pair programming and solo programming later modules [12]). The compromise is a mixture of independent and paired assessment (as suggested by Jacobson and Schaefer [11],) encouraging the development of both pair and individual programming skills.

Making the review process individual eliminates the problems associated with group and pair work as the student is working alone. Research has been conducted into individual code reviews and their improvement on solo programming and Humphrey provides some evidence that students using the PSP (which includes code review) show improved development skills [10]. This suggests individual reviews may be well suited to solo programming assignments. However, PSP itself is a formal documentation heavy technique focused on recording metrics (development time, lines of code, defects etc.,) of a program in order to gauge its size and complexity as a basis for future estimation [13]. The lengthy documentation is a cause of resistance amongst some students and training in the process is required [13].

The aim of this research is to ascertain whether individual code reviews based on checklists (like those used in PSP [10] and during formal code inspections in industry [14],) with minimal reporting can be used to improve solo programming. This will allow students to systematically check their work as they would have done for pair programming and complement the pair programming approach currently used within the module.

A preliminary study was conducted on first year students enrolled on BSc. Software Engineering and BSc. Games Development at Swansea Metropolitan University. This is a small group (around 25 in total,) who study C and C++ programming in their first year. For the experiment the students were split into two groups (a test group and control group,) with alternating students being picked from a sorted list based on an earlier assignment. The test group was provided with a checklist (based on widely available code reviews [15] and teaching material,) along with some brief training. The students were also tasked with recording which of the points helped them to identify defects in their code (though simple annotation of the list rather than a formal reporting process). Unlike the test group the control group were not provided with the checklist. The average mark of the group on the assessment was used to quantify the results (as used by Brykczynski [15]), accompanied by a survey which provides an insight into the student's perception of the process for future refinement (as used by Hundhausen *et al* [2]).

The results shown an increase in performance but due in part to the small sample size this not statistically significant. The survey results are more encouraging suggesting that the students who undertook the code review believed the process to be beneficial and that undertaking the review had improved the programs they had produced. Additionally the majority of students reported they would use code reviews for future assignments.

In conclusion it remains unclear whether code reviews can be used as an additional component to first year assignments to improve program quality. Perhaps as software engineering texts suggest quality cannot be built in at the end [14]. However from a pedagogical perspective students have found the process beneficial they feel their programs have improved suggesting they have found errors and omissions in their code and corrected them. They have also been reminded of things they have learned and can apply and in some cases gained additional knowledge. An analogy can perhaps be made with proof reading, observations suggest only those students who finish their reports in good time will proof-read (this has also been observed in other studies [16]). Similarly few students will proofread their code to identify errors/omissions once the code compiles and passes rudimentary testing it is submitted. Code reviews may provide a mechanism for encouraging students to proofread code prior to submission.

Current studies include a number of potential improvements, notably the code review now forms part of the submission and has associated marks providing students with a crude incentive for students to undertake the review. The submission takes the form of a table where students to indicate which elements they found useful providing a mechanism for improving the review in future years.

The code review is being evolved on a topic by topic basis. This provides students with a reference for best practice and a review checklist for the staged formative assessment. Common errors from which can be added to the checklist. This incremental approach provides an opportunity to train students in the review process motivated by research stating that peer and individual assessment in higher education requires training and academic maturity amongst the students [17] and the results of the earlier survey. If exposing students to the peer review process earlier in the year can be shown to provide them with a framework for assessing their code (and making improvements,) it may be possible to extend the checklist into a means for self or peer assessment. Future studies will include all students studying introductory programming modules allowing for a deeper analysis of the results.

[1] J. Sheard, S. Simon, M. Hamilton, and J. Lonnberg, "Analysis of reserach into teaching and learning programming," presented at 5th International workshop on Computing education research, Berkeley, CA, USA, 2009.

[2] C. Hundhausen, A. Agrawal, D. Fairbrother, and M. Trevisan, "Intergrating Pedagogical Code Reviews into a CS 1 Course: an Emperical Study," *ACM SIGCSE*, vol. 41, pp. 291-295, 2009.

[3] J. K. Huggins, "Engaging Computer Science Studetns through Cooperative Education," *ACM SIGCSE*, vol. 41, pp. 90-94, 2009.

[4] D. A. Trytten, "A Design for Team Peer Code Review," *ACM SIGCSE*, vol. 37, pp. 455-459, 2005.

[5] C. McDowell, L. Werner, H. E. Bullcock, and J. Fernland, "The effects of pair-programming on performance in an introductory programming course," presented at ACM Special Interest Group on Computer Science Educaton Technical Symposium, Cincinnati, Kentucky, 2002.

[6] C. McDowell, L. Werner, H. E. Bullcock, and J. Fernland, "Pair Programming Improves Student Retention, Confidence and Program Quality," *Communications of the ACM*, vol. 39, pp. 90-95, 2006.

[7] L. Williams and R. L. Upchurch, "In Support of Student Pair-Programming," *ACM SIGCSE Bulletin*, vol. 33, pp. 327-331, 2001.

[8] J. K. Doyle, "Improving performance and retention in CS1," *Journal of Computer Science in Colleges*, vol. 21, pp. 11-18, 2005.

[9] Y. Wang, Y. Li, M. Collins, and P. Liu, "Process Improvement of Peer Code Reivew and Behaviour Analysis of its Participants," *ACM SIGCSE*, vol. 40, pp. 107-111, 2008.

[10] W. S. Humphrey, "Finding Defects," in *Introduction to the Personal Software Process*. New York, USA: Addison Wesley Longman Inc, 1997, pp. 157-174.

[11] N. Jacobson and S. K. Schaefer, "Pair Programming in CS1: Overcomming Objections to its Adoption," *ACM Special Interest Group on Computer Science Education*, vol. 40, pp. 93-96, 2008.

[12] B. Simon and B. Hanks, "First-year students' impressions of pair programming in CS1," *Journal on educational resources in computing*, vol. 7, pp. 5:1-5:28, 2008.

[13] S. A. Bloch, "Scheme and Java in the first year," *Journal of Computing in Small Colleges*, vol. 15, pp. 157-165, 2000.

[14] I. Sommerville, "Verification and Validation," in *Software Engineering*, 8th ed. Essex, England: Pearson Education Limited, 2007, pp. 516-136.

[15] B. Brykczynski, "A Servey of Software Inspection Checklists," *ACM SIG SOFT Software Engineering Notes*, vol. 24, pp. 82-89, 1999.

[16] T. J. Gambell, "University Education Students' Self-Perceptions of Writing," *Canadian Journal of Education*, vol. 16, pp. 420-433, 1991.

[17] S. Fallows and B. Chandramohan, "Multiple Approaches to Assessment: Reflections on the use of Tutor, Peer and Self Assessment," *Association for Learning Technology Journal (ALT-J)*, vol. 9, pp. 26-37, 2001.