Peer-to-peer programming versus individualised programming: The real world

Desmond W. Govender, University of KwaZulu-Natal, South Africa
T.P. Govender, Durban University of Technology, South Africa

ABSTRACT

Does programming alone depict the real world scenario? It is often said, 'In programming, One is the loneliest number'. Addressed in this paper is a strategy referred to as 'peer-to-peer programming', and the focus is within an object-oriented programming (OOP) paradigm. The paper explores options available to students at secondary and university levels who are engaged in individualised programming, and potential opportunities associated with peer-to-peer programming which resemble programming in the real-world scenario. Most programmers learn to program as individuals and then are faced with the real world, programming in a team. This research was based on confirmed observation, analysis and assessment against published data and information extracted on peer-to-peer programming and its influence on contemporary instruction and comprehension of computer science programming concepts. The research was conducted within the parameters of the programming information technology curriculum at three secondary and two university-level academic institutions. Gaining the perceptions of peer-to-peer programming was arrived at through using threshold concepts in Computer Science and an interpretive paradigm. Two critical questions were posed:

- What are learners' experiences of programming alone and programming with a partner?
- How does peer-to-peer programming enhance programming in an OOP paradigm?

Based on assessment and analysis of the data, our recommendation is to create an environment of peer-to-peer programming in an OOP setting which is similar to programming in a team in the real- world scenario. The benefits are not only better problem solving and better programmers but also the enhancement of good social skills.

Keywords: peer-to-peer learning, learning programming, collaborative learning, object-oriented programming

INTRODUCTION

Many will agree that programmers have a reputation for not being social. Yet, we have to work in teams all the time, and use highly social development processes. In schools and universities we learn programming as a 'Lone Wolf' developer, while in the real business world (software houses), we are required to program in a team. Over the course of the last half century much discussion has continued as to the most beneficial and effective means of presenting the elements of Computer Science programming and the relevant associated programming languages. A single paradigm embodied in a single language is deemed by Roy and Haridi (2003) as the most suitable application for programming instruction and training, while Cooper, Dann and Pausch (2003) believe in an 'objects-first application'. Furthermore, Howe, Thornton and Weide (2004: 291) contend that the approach with the most impact is the 'object- oriented' (OO) and 'component-first' approach. Research by Govender (2006: 19) demonstrated that there 'exists tension between procedural paradigm and OO paradigm', although an OO approach to programming is 'not a good starting-point for introducing students to the basic concepts of programming', according to Ismail, Ngah and Umar (2010: 216).

Contemporary data and information indicates that examination results are cause for concern, and the percentage rate of first-year students abandoning the programme is increasing at a staggering rate (Department of Higher Education and Training, 2012). Many believe that this can be attributed to an improper utilisation of available resources and issues associated with student learning and comprehension.

A collaborative learning framework has been implemented to counter the abandoning and failure rate of first-year students (Hakizimana & Jürgens 2012; Letseka & Maile 2008; Horn, Collier, Oxford, Bond jun. & Dansereau, 1998). The premise of this framework is structured on the collaborative efforts of each individual (Horn et al., 1998). This study's intent was to establish if a collaborative learning framework is a practical pedagogical tool for instruction and comprehension of an introductory programming curriculum at secondary schools and university-level institutions within KwaZulu-Natal and to determine to what extent this collaboration will develop skills in programmers that are needed when programming in a team. Its focus was to decide if collaborative Pair Programming (PP) has a definitive influence on those individuals' learning, and if it is deemed a practical application for Information Technology (IT) instructors to utilise. More intensive observation would also reveal the views and opinions of IT instructors in KwaZulu-Natal and their application of PP as a means towards achieving the instructional and comprehension methods regarding the OO paradigm.

In introductory programming courses at university level, Microsoft C# and Java were utilised as primary programming languages. Programming problems were solved with a specific programming language and within a particular programming paradigm. Enrolled students at all education levels have the ability to excel at programming individually, but in an organisational setting the environment is more of a collaborative effort with other individuals. This can present a myriad of problems for individuals who are used to solving problems individually.

Studies were crafted to assess PP usage and its benefits in association with the instruction and comprehension of an introductory programming curriculum (Mentz, van der Walt & Gorsen, 2008; Nosek 1998; Williams, Robert, Cunningham & Jeffries, 2000). Some of the advantages of PP noted are high program quality, efficient time management of programs, enhanced comprehension of the programming methodology, elevated course completion percentages and examination performance rates (Preston, 2005). In North West Province, South Africa, successful PP application of instructor training was attained, according to Mentz et al. (2008). Contemporary data and information gathered from research by Breed et al. (2013) showed that elevated productivity is attainable by means of meta-cognitive skills during PP. The nature of the data from PP usage served as an impetus for the implementation of this study.

The objective was to provide learners and educators at schools and tertiary institutions with a strategy that they could adopt when teaching and learning introductory programming courses.

The theoretical and conceptual framework for this study comprised threshold concepts in Computer Science. A threshold concept can be considered as akin to a portal, opening up a new and previously inaccessible way of thinking about something. It represents a transformed way of understanding, interpreting or viewing something without which the learner cannot progress. Threshold concepts are those which might be used to organise the educational process, and are likely to be transformative, integrative, irreversible, and potentially troublesome for students, and are often boundary markers (Cousin & Meyer, 2006). The interpretive paradigm was chosen for this study because it is concerned with descriptions that produce deep understanding and emphasise interpretation of data from educators and learners. It is within the boundaries of this theoretical framework that the following research questions were answered:

- What are learners' experiences of programming alone and programming with a partner?

And

- How does peer-to-peer programming enhance programming in an object-oriented programming (OOP) paradigm?"

OVERVIEW OF THE LITERATURE

Benefits of the use of PP

Many sectors of the software development industry embrace PP, and it is utilised as a training application (Nagappan et al., 2003). Its usage is widespread in the workplace, according to Chong and Hurlbutt (2007). Employees do not receive the benefit of a collaborative effort when they program individually, as they are not exposed to alternative ideologies of resolution, communication and inspiration. Working in a collaborative environment fosters self-esteem, builds on one's confidence and serves as a springboard for further advancement, according to Williams et al. (2000).

Elevated knowledge utilisation resulting from meta-cognitive skills implemented during PP was deduced from data and information research conducted by Breed et al. (2013). The primary element that initiates difficulty in comprehending programming and coding is the 'inactive involvement' of individuals engaged in programming tutorials, according to Ismail, Ngah and Umar (2010: 128). PP was overwhelmingly preferred over individual programming, as individuals were prone to making more mistakes, according to Nosek (1998).

Comparable data and information were revealed in our study. Braught, Eby and Wahls (2008) released information that indicated that students who had low Mathematics standardised test scores demonstrated improvements in individual programming skill when exposed to PP. In addition, drop-out percentages lowered and academic achievement rose. Nagappan et al. (2003) contend that PP was instrumental in better course completion rates.

A positive student experience at the outset of any introductory programming curriculum greatly enhances the probability of continuing on the course track in Computer Programming (McDowell, Werner, Bullock & Fernald, 2002). Not limited to just males, female programmers collaborating excelled (Werner, Hanks & McDowell, 2004). Research data and information from Carter (2006: 29) revealed that a high percentage of females refrain from entering the field of Computer Science, citing their wish to undertake a more 'people-oriented' field of study.

Females enrolled in the various grades/levels of higher learning found PP to be of great benefit and enjoyment in their IT and programming class assignments, according to Liebenberg, Mentz and Breed (2012). Female productivity was far greater when in a collaborative environment, with a higher grade of finished product, completed in a time-efficient manner.

Such a climate of productivity and efficiency elevated their confidence and fostered an elevated interest in IT careers (Berenson, Slaten, Williams & Ho, 2004).

Nearly 20% of Computer Science students in the United States of America are female (Zaidman 2011). PP is a 'people-oriented' strategy; therefore employing techniques of PP augurs well for the inclusion of more females in undergraduate courses and subsequently in the workplace. Females made up nearly 33% of the sample population in this study.

Improved understanding of programming principles

Utilising PP can enhance an individual's comprehension of programming principles, contemporary research confirms (Prottsman, 2014). Assembling groups for collaborative efforts permitted these individuals to demonstrate further the knowledge they acquired, progressing their knowledge of the IT spectrum, according to Barker, Garvin-Doxas and Roberts (2005). Similar findings were revealed by Tillema and Van der Westhuizen (2006), with groupings of individuals promoting concern and attentiveness towards resolving issues. Research by Preston (2005) indicated that PP was an attractive means to employ to have students learn basic programming syntax and principles.

McDowell et al. (2002) contend that results for PP sections were highly elevated in comparison to those from non-pairing classes, and collaborative efforts demonstrated higher-quality programs. Nagappan et al. (2003) contend that improved course completion percentages were a demonstrated advantage of PP. Bain (2004) suggests that 'undergraduates learn most effectively when they can work collaboratively with other learners to grapple with problems'.

This study found a similar result. Educators must encourage and stimulate individuals that are learning to expand further their realm of knowledge, according to Berglund, Box, Eckerdal, Lister and Pears (2008). Investigative research suggests that the social interactions of students may be an indicator towards lowering the percentage of individuals that drop out of the curriculum, according to Kinnunen and Malmi (2006).

PP offers an opportunity for individuals to enhance their own comprehension and optimise such knowledge in a collaborative arena with other individuals (Sextro, 2012). Programming is optimised as a collaboration and not as a solitary occurrence. Observations of this study in KwaZulu-Natal revealed students in a collaborative effort conversing in their mother tongue when confronted with a major obstacle or hindrance. Use of their shared mother tongue put everyone on the same plane of thought, which is conducive towards finding resolution.

Preston (2005: 41) cited the following benefits to the use of PP: 'Higher quality of programs; Decreased time to complete programs; improved understanding of the programming process; improved course completion rates; improved performance on exams'. Mentz et al. (2008: 250) concluded that 'the experimental group not only outperformed the control group but there were also were fewer drop outs'.

Cockburn and Williams (2001) summarised the significant benefits of PP as follows:

- Errors are discovered at key entry instead of during quality assurance applications or in the field (continuous code reviews)
- The end defect content is statistically lower (continuous code reviews)
- The designs are better and code length is shorter (ongoing brainstorming and pair relaying)
- The team solves problems faster (pair relaying)
- The people learn significantly more about the system and software development (line-of-sight learning)
- The project ends up with multiple people understanding each piece of the system.

Benefit to inexperienced and experienced programmers

In order to obtain diverse results, research conducted by Padmanabhuni, Tadiparthi, Yanamadala and Madina (2012) assembled individuals in pairs across a vast platform of knowledge. Results from their assessment and analysis of their research indicated that PP is an efficient technique in programming that provides an opportunity for individuals to expand further their capabilities and knowledge.

Comments from programmers with verified experience have stated that they have found benefit in being paired with junior programmers (Lang & Ottinger, 2011: 3-5). Some of the benefits included:

- As a new hire in a pairing environment, you don't spend week one (or month one) sitting and reading out-of-date documentation or fearing a code base that you can barely begin to understand on your own. Instead, you get to jump right in and wet your feet with live production code. The rest of the team doesn't resent having to take time out from 'their' work to answer your endless questions about the system -- they can instead work with you directly, because that's how the team has chosen to work.
- We love learning new things about software development. We think we're pretty good at programming, yet rarely a day goes by when we don't learn something new and significant -- even from the most junior programmers on the team.

- If you're the team's rock star, pairing can give you mentoring and teaching opportunities that you've never had before, plus the respect you deserve. Invariably, a great programmer on any team becomes revered by the team. If you have the skills alone, you have the skills paired too.
- If you are the weakest player on the team, you will find that pairing gives you an opportunity to learn from your teammates. In addition, as the partner shares the keyboard and ensures that you're doing test-first work, you will find that it's harder to make a mistake that gets through to integration. You have a safer working/learning environment.
- When you are tired, frustrated, less well, hungover, underslept, low on biorhythms or feeling unlucky, you are far more likely to stay engaged and productive if you are pairing. Partners look out for you. Your worse days pairing won't look like your worse days as a solo programmer.

While this study only studied students on an equal academic level, it should be noted that educators should give due consideration to pairing programmers from various academic year groups. However, this was beyond the parameters of this study.

Increased social development and enjoyment of programming

A critical benefit of PP, according to Pikkarainen, Haikara, Salo, Abrahamsson and Still (2008), was a high rate of interaction amongst those participating in the collaborative effort. Observations recorded during research indicated in excess of 250 verbal interactions per PP hour, and both partners collaborated on 93% of their tasks. Such collaborative efforts stimulate productivity, which counters the notion that talented individuals can resolve issues on their own and without interaction with others (Cockburn & Williams, 2001). Cockburn and Williams' research also revealed more confidence and heightened harmony amongst those who were interacting. Research conducted by Liebenberg (2010) confirmed the findings of Cockburn and Williams (2001) of greater fulfilment and vision of programming. Lang and Ottinger (2011) noted the following:

> …having a whole team in a room can be noisy and distracting, while a focused pair can more easily block out distractions than an individual and that people are also less likely to interrupt a pair deep in work and conversation than an individual sitting alone.

An elevated verbal interaction was observed by Pikkarainen et al. (2008) and Lang and Ottinger (2011) indicated that the interaction between two programmers eliminated interruptions imposed by other individuals.

Costs related to software development and programming

Roughly 60% of software projects which are initiated do not come to fruition (Molokken & Jorgensen, 2003). Programmers are often viewed by managers as a prized commodity, who command a high price – which is a reason PP principles are not implemented in many instances. Cockburn and Williams (2001: 95) contend that 'the development cost for the benefits of PP is not the 100% that might be expected, but is approximately 15% and this is repaid through shorter and less expensive testing, quality assurance and field support'.

Lax planning, insufficient engineering, untimely decisions and poorly calculated estimates serve as reasons for software project failures, according to Galorath and Evans (2006). Wrongly calculated estimates are the core of virtually every software project failure, contend Molokken and Jorgensen (2003).

PP can lower the amount of the time spent by student programmers on academic software projects, according to our study. Macgregor (2007: 1) states that '40% of South African students drop out of tertiary institutions in their first year of study'. A major issue is the elevated rate of individuals dropping out, which would suggest that PP would increase individual passing percentages, while at the same time lowering costs. Success in contemporary programming in conjunction with utilisation of PP can be attributed to where programmers interactively engage in group programming, designing, and testing and assume new responsibilities with other group members. In many instances collaborative efforts are implemented in the review of software design prior to full integration. The Chrysler Comprehensive Compensation system is a prime example of the implementation of PP, according to Anderson, Beattie and Beck (1998).

The practical application of PP makes it viable for institutions of higher education to adopt (Salleh, Mendes, Grundy & Burch, 2010). If viable in the industrial field and with students being prepared to enter the global marketplace, should there not be a mandate for industry standards of work ethics and practices? Research conducted by McDowell et al. (2002) indicated that scores from PP were noticeably elevated in comparison to scores of individuals that did not participate in PP, serving as valid proof that collaborative efforts significantly enhance software program quality.

As academic achievement was attained, the percentages of individuals that withdrew from the program decreased. Improved course completion rates were derived from use of PP

(Nagappan et al., 2003). As with many undergraduates, learning can be a troublesome issue, but grades improve when they engage in a collaborative effort (Bain, 2004).

The use of PP enhanced cognitive skills of individuals and optimised their own productivity, according to Breed et al. (2013). Cost-effectiveness and efficiency are realised in organisational, industrial and academic frameworks when the percentage of individuals withdrawing from the curriculum is reduced and overall productivity increases. In conjunction with monetary gain through implementation of PP, are the successful launches of software coding. Profit is increased, downtime is reduced and in an academic environment government funding increases.

Time constraints often hinder student-teacher interactions in settings that foster large assemblies of individuals. This necessitates that educators seek other means of achieving face-to-face consultations with students with computer-mediated models of communication (Field, 2005).

A University of the Witwatersrand study revealed that person-to-person consultation with individuals in large assemblies does not foster learning in collaborative efforts, and promotes more individuality and one-on-one competition, counter to the ideology of PP. Such vast numbers, in this case 600 students, of individuals in a large learning arena (lecture hall) impose an undue burden on the educator and stymies the intent of fostering learning and knowledge (Thatcher, 2007).

The University of South Africa (UNISA) is a distance-learning/video-conferencing learning institution that initiated intervention strategies to support its programmes (Macgregor, 2007) in 2007, investing nearly R50 million. Utilised at regional education facilities, peer-to-peer learning was an application designed to elevate the percentage of individuals passing the curriculum and eradicating the high withdrawal rate of students from the programme.

Salleh et al. (2010) demonstrated by means of anecdotal and empirical data and information that PP resulted in major strides in the performance of introductory programming courses in Computer Science. Initial qualitative evidence leads one to believe that PP improves not only learning amongst introductory programming individuals, but assists educators in maximising time spent in consultations with students.

The second principle from Ramsden (1992) for effective academic instruction relates to understanding how individuals comprehend matters presented before them; the strategies employed were a conversational framework, consultation and negotiation processes.

In lieu of consultation and negotiations, PP would be employed as an applicable strategy by those in academia to optimise their relationship with students in the instruction of programming in a collaborative manner between individuals. Research for this study was motivated by the need for relevant and necessary support for individuals learning programming and coding. Achieving such support is critical, and PP offers one such solution to complementing traditional lecturer/student consultation and reduces the percentage of individuals withdrawing from the IT curriculum.

METHODOLOGY

The participants in this research consisted of 60 Grade 11 and 45 Grade 12 IT learners from three secondary schools, and 75 first years IT students from two universities. It was deemed that two or more data collection methods were necessary to lend credibility to and substantiate this study. Methods employed to assemble data included: (a) personal interviews, (b) observations, and (c) questionnaires. Observations were made of collaborative pairs, and participants were required to respond to pre- and post-pairing questionnaires. Prior to and after the intervention, educators were interviewed.

Data Analysis and Interpretation

Information and data gathered from questionnaires and observation sheets were assembled as numerical data in a spreadsheet format that enabled statistical representation in percentages and graphs. Results of data mining were assembled into relevant categories in conjunction with the theoretical framework, steered by the questions posed.

The intent of the study was to determine the application of PP as a means of supporting the introduction and demonstration of an introductory programming curriculum, and to engage an appropriate and beneficial strategy for educators and administrators to adopt.

RESULTS

The findings which emanated from the study are now summarised according to each of the research questions.

What are learners' experiences of programming alone and programming with a partner?

To ascertain the individuals' perceptions of programming, pre- and post-test questionnaires were administered. Questions posed to individuals inquired as to their views towards the benefits of PP. Responses indicated that there was a strong sentiment towards PP having value. The overall view of participants on the questionnaire was that 'it was an indispensable tool, critical in assisting with the comprehension and understanding of various programming concepts'. They believed that PP was 'instrumental in enhancing the programming abilities and was a positive strategy necessary for success'. With the majority of responses to the questionnaire being favourable, educators expressed a high level of confidence in the implementation of PP concepts.

PP was perceived, from responses, as 'easy to implement and adopt' and embraced by individuals and educators. Individuals expressed their confidence in PP and saw it as an appropriate avenue for the resolution of an issue with which they were presented. Individuals that often preferred individual assignments expressed an approval of a paired assignment in a collaborative effort. Although PP was readily embraced in this study, it was found that it was not fully implemented at the secondary level and at universities as reflected in our observations.

If academia were to fully employ PP, it would heighten programming concepts learning, according to this study. Many participants sincerely believe that PP is a suitable choice for 'lending support in a learning environment'. Data confirm the long-standing belief that pairing individuals in a collaborative effort sustains quality software production and fortifies individuals' quest for knowledge.

Research data indicated that enhanced forms of enjoyment were experienced by individuals when paired with another individual in lieu of a solitary endeavour. In their responses male programmers indicated that they 'felt more productive when paired with a female programmer'. Results from post-test data and information indicate that individuals were confident with resource availability, paired partners and educator support.

How does peer-to-peer programming enhance programming in an OOP paradigm?

Results revealed that individuals enjoyed addressing potential strategy avenues with the paired partner instead of their instructor in a PP environment. Individuals only turned to their instructor as a last resort, first exhausting all other options with their partner, although some individuals expressed hesitancy in approaching an instructor to find resolution of the issue.

The following benefits were derived from the learners' interaction with PP:

(a) enhanced communication (collaboration)
(b) interactive participation with peers
(c) greater accessibility of resources
(d) a supportive and favourable setting
(e) enhancement of collaborative learning.

A collaborative effort has the advantage of allowing the learner to discuss their experiences with their paired partner prior to approaching their instructor, which fosters independent learning and problem solving.

Individuals found an ease in collaborating with a paired partner from the same linguistic group or mother tongue. Cultural background has a large influence and impact in fostering an amicable relationship amongst collaborators. Interactive participation is a mutual benefit within a PP scenario. It fosters a climate of assistance and support in search of errors and omissions, and is a learning experience which may be built upon.

The findings of this study showed that the learners found PP to be an easy, efficient and enjoyable way to learn problem-solving techniques.

OVERVIEW OF RESULTS

This study ascertained that implementing PP achieves the following:

- PP contributes towards motivating students to complete a programming task or even learn a new programming concept, and the probability of successful compilation and completion is greatly increased by working in pairs.
- Paired programmers can develop strong relationships of friendship that go beyond the programming task assigned to them, and the continuous discussion of a programming task or concept makes for more sociable and better programmers.
- The process of PP encouraged academically weaker students to develop their programming ability by acquiring programming skills from a peer in an informal setting, and similarly academically strong students acquired the humility of learning to keep their ego in check and learnt alternate programming solutions.
- PP can afford the educator more time and create a more conducive, stress-free learning environment – whereas previously the educator was inundated with programming queries, with PP the query is discussed within the pair and most often solved within the paired partnership.

- Programming ‘mistakes’ or bugs are discussed within a pair and not in front of the entire class.
- Pair partners look out for each other and vehemently support their program when confronted by detractors.
- Not all learning and teaching environments are conducive to implementing PP. For effective PP experiences educators need to rearrange the classroom and computer facilities.
- Educators must support the use of PP in the classroom; without such support, PP is doomed from its inception.
- Sometimes it may be necessary to halt PP implementation and allow for individual programming or even a completely different activity, then regroup the pairs or even change paired partners.
- It is essential that when individuals are paired in collaboration, educators break down any barriers when pairing individuals together. This provides for a more conducive and less competitive environment, and in most instances leads to fostering a spirit of cooperation.

All of the above benefits are crucial in a business environment in the real world. To advance further the benefits and attributes of PP, it is strongly suggested that academic institutions become more actively involved in collaborative and PP techniques. With proper instruction and training for educators and administrators, PP will then be properly introduced and demonstrated to those individuals that wish to advance further their career path in IT and enhance their skills through the utilisation of collaborative efforts.

CONCLUSION

Individual programming has always been the norm in schools and universities. Traditional teaching methods were premised on the ‘all-knowledgeable’ educator who ‘spoon-fed’ a learner. With advances in contemporary technology such as the Internet, video conferencing, cell phones and computers, these elements are instrumental in the promotion and utilisation of PP in the classroom as a means to resolving issues associated with the negative elements (especially problem solving skills) associated with contemporary programming. Using PP will ensure that students are being prepared for the real world where they will be required to program as a team.

It is the view of the authors that if recommendations culled from this study are properly implemented, PP can be of great benefit in the classroom environment. PP is an adjunct to

the traditional classroom environment of teacher-student learning. A traditional classroom is where teacher and students connect, and in most instances is not a setting for collaboration. Introduction of PP induces group participation and collaboration. Such collaboration stimulates creative thinking, which is more applicable towards finding resolution of programming issues.

With annual exposure to PP in an academic environment, PP is positioned to provide those individuals learning programming with an opportunity to get one step ahead of others in preparation to enter the workforce in years to come.

REFERENCES

Anderson, A., Beattie, R. & Beck, K. (1998) 'Chrysler Goes to Extremes' Distributed Computing 54(1) pp.24-28.

Bain, K. (2004) What the Best College Teachers Do. Cambridge, MA: Harvard University Press.

Barker, L.J., Garvin-Doxas, K. & Roberts, E. (2005) 'What Can Computer Science Learn from a Fine Arts Approach to Teaching?' Paper presented at the SIGCSE St. Louis, Missouri, USA. ACM Science Bulletin 37(1) pp.421-425.

Berenson, S.B., Slaten, K.M., Williams, L. & Ho, Chih-Wei. (2004) 'Voices of women in a software engineering course: reflections on collaboration' Journal on Educational Resources in Computing (JERIC) 4(1) pp.5-12.

Berglund, A., Box, I., Eckerdal, A., Lister, R. & Pears, A. (2008) 'Learning educational research methods through collaborative research: the PhICER initiative' Proceedings of the tenth conference on Australasian computing education 78 pp.1-8.

Braught, G., Eby, L.M. & Wahls, T. (2008) 'The effects of pair-programming on individual programming skill' ACM SIGCSE Bulletin 40(1) pp.200-204.

Breed, B., Mentz, E., Havenga, M., Govender, I., Govender, D., Dignum, F. & Dignum, V. (2013) 'Views of the use of self-directed metacognitive questioning during pair programming in economically deprived rural schools' African Journal of Research in Mathematics, Science and Technology Education 17(3) pp.206-219.

Carter, L. (2006) ‘Why students with an apparent aptitude for computer science don’t choose to major in computer science’ ACM SIGCSE Inroads Bulletin 38(1) pp.27-31.

Chong, J. & Hurlbutt, T. (2007) ‘The social dynamics of pair programming’ Proceedings of the 29th International Conference on Software Engineering (ICSE 07) pp.354-363. IEEE CS Press.

Cockburn, A. & Williams, L. (2001) ‘The costs and benefits of pair programming, in eXtreme Programming and Flexible Processes in Software Engineering—XP2000’ pp.309-326. Paper presented in Cagliari, Sardinia, Italy.

Cooper, S., Dann, W. & Pausch, R. (2003) ‘Teaching objects-first in introductory computer science’ Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education pp.191-195.

Cousin, G. & Meyer, J. (2006) ‘Threshold concepts, troublesome knowledge and emotional capital. Overcoming barriers to student understanding: Threshold concepts and troublesome knowledge’ Proceedings of the 12th Improving Student Learning Conference. Oxford: Oxford Centre for Staff and Learning Development (OCSLD) pp.53-64. http://www.brookes.ac.uk/services/ocsld/isl/isl2004/ abstracts/conceptual_papers/ISL04-pp.53-64-/_and-et-al.pdf (Accessed 10 August 2014).

Department of Higher Education and Training. (2012) The Green Paper for Post School Education and Training. South Africa. http://www.dhet.gov.za/Linkclick.aspx?fileticket=w0qJyEiFVYQ%3d&tabid=189 &mid=483 (Accessed 14 May 2014).

Field, R.M. (2005) ‘Favourable conditions for effective and efficient learning in a blended face-to-face/ online method’ Asilite 2005: Balance, Fidelity, Mobility: Momentum 4-7 December 2005. Queensland University of Technology. https://eprints.qut.edu.au/18100/lc18100.pdf (Accessed 20 May 2015).

Galorath, D. & Evans, M.W. (2006) Software sizing, estimation, and risk management: when performance is measured performance improves. Boca Raton, Florida: CRC Press.

Govender, I. (2006) Learning to program, learning to teach programming: pre- and in-service teachers' experiences of an object-oriented language. Unpublished PhD thesis, UNISA, Pretoria, South Africa.

Hakizimana, S. & Jürgens, A. (2012) 'The Peer Teaching/ Learning Experience Initiative: Feedback and Student Perception' 6th Annual Teaching & Learning Conference pp.25-37. September 2012, University of KwaZulu-Natal, Howard College Campus, South Africa.

Hanks, B., McDowell, C., Draper, D. & Krnjajic, M. (2004) 'Program quality with pair programming in CS1' Paper presented at the 9th Annual Conference on Innovation and Technology in Computer Science Education, 28-30 June 2004.

Horn, E. M., Collier, W.G., Oxford, J.A., Bond Jr, C.F. & Dansereau, D.F. (1998) 'Individual differences in dyadic cooperative learning' Journal of Educational Psychology 90(1) pp.153-159.

Howe, E., Thornton, M. & Weide, B.W. (2004) 'Components-first approaches to CS1/CS2: principles and practice' Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education pp.291-295. New York: ACM.

Ismail, M.N., Ngah, N.A. & Umar, I.N. (2010) 'Instructional Strategy in the teaching of computer programming: a need assessment analyses' The Turkish Online Journal of Educational Technology 9(2) pp.125-131.

Kinnunen, P. & Malmi, L. (2006) 'Why students dropout CS1 course?' Proceedings of the Second International Workshop on Computing Education Research pp.97-108. New York: ACM.

Lang, J. & Ottinger, T. (2011) 'Pair Programming Benefits Two Heads Are Better than One' https:// pragprog.com/magazines/2011-07/pair-programming-benefits (Accessed 14 May 2014).

Letseka, M. & Maile, S. (2008) 'High University Drop-out Rates: A Threat to South Africa's Future' Policy Brief (March). Pretoria: Human Sciences Research Council.

Liebenberg, J. (2010) Secondary school girls' experiences of pair programming in information technology. (M.Ed. dissertation) Potchefstroom: North-West University, South Africa.

Liebenberg, J., Mentz, E. & Breed, B. (2012) ‘Pair programming and secondary school girls’ enjoyment of programming and the subject Information Technology (IT)’ Computer Science Education 22(3) pp.219-236.

Macgregor, K. (2007) South Africa: Student drop-out rates alarming. http://www.universityworldnews. com/article.php?story=20071025102245380&mode=print (Accessed 29 May 2014).

McDowell, C., Werner, L., Bullock, H. & Fernald, L. (2002) ‘The effects of pair-programming on performance in an introductory programming course’ ACM SIGSE Bulletin 34(1) pp.38-42.

Mentz, E., Van der Walt, J.L. & Goosen, L. (2008) ‘The effect of incorporating cooperative learning principles in pair programming for student teachers’ Computer Science Education 18(4) pp.247-260.

Molokken, K. & Jorgensen, M. (2003) ‘A review of software surveys on software effort estimation. Paper presented at the Empirical Software Engineering, 2003’ ISESE 2003 Proceedings pp.223-230. 2003 International Symposium on Empirical Software. 30 September-10 October 2003.

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C. & Balik, S. (2003) ‘Improving the CS1 experience with pair programming’ Proceedings of ACM Special Interest Group in Computer Science Education (SIGCSE 2003) pp.359-362. Reno, NV: ACM SIGCSE.

Nosek, T. (1998) ‘The case for collaborative programming’ Communications of the ACM 41(3) pp.105-108.

Padmanabhuni, V.V.K., Tadiparthi, H.P., Yanamadala, M. and Madina, S. (2012) ‘Effective Pair Programming – An Experimental Study’ Journal of Emerging Trends in Computing and Information Science 3(4) pp.471-479.

Pikkarainen, M., Haikara, J., Salo. O., Abrahamsson, P. & Still, J. (2008) ‘The impact of agile practices on communication on software development’ Journal of Empirical Software Engineering 13(2) pp.303-337.

Preston, D. (2005) ‘Pair programming as a model of collaborative learning: a review of the research’ Journal of Computing Sciences in Colleges 20(4) pp.39-45.

Prottsman, K. (2014) ‘Computational thinking, Coding - 3 best practices for pair programming’ International Society for Technology in Education (ISTE) https://www.iste.org/explore/articledetail?articleid=221 (Accessed 10 June 2014).

Ramsden, P. (1992) Learning to teach in higher education. London: Routledge.

Roy, P. & Haridi, S. (2003) ‘Teaching programming broadly and deeply: The Kernel language approach’ In L. Cassel & R. Reis (Eds.) Informatics Curricula and Teaching Methods 117 pp.53-62.

Salleh, N., Mendes, E., Grundy, J. & Burch, J. (2010) ‘An empirical study of the effects of conscientiousness in pair programming using the five-factor personality model’ Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE2010). Cape Town, ACM Press.

Sextro, J. (2012) ‘Secrets of Pair Programming Perfection’ http://www.Devx.Com/Enterprise/Seven- Secrets-Of-Successful-Pair-Programming.html (Accessed 10 February 2015).

Thatcher, A. (2007) ‘Using the World Wide Web to support classroom lectures in a psychology course’ South African Journal of Psychology 37(2) pp.348-353.

Tillema, J. & van der Westhuizen, G.J. (2006) ‘Knowledge construction in collaborative enquiry among teachers’ Teachers and Teaching: theory and practice 12(1) pp.51-67.

Werner, L., Hanks, B. & McDowell, C. (2004) ‘Pair-programming helps female computer science students’
Journal on Educational Resources in Computing (JERIC) 4(1) pp.1-4.

Williams, L.K., Robert, R., Cunningham, W. & Jeffries, R. (2000) ‘Strengthening the case for pair programming’ IEEE Software 17(4) pp.19-25.

Williams, L., Wiebe, E., Yang, K.F.M. & Miller, C. (2002) ‘In support of pair programming in the introductory computer science course’ Computer Science Education 12(3) pp.197-212.

Zaidman, Marsha. (2011) ‘Inspiring future female scientists’ Paper presented at the 41st ASEE/IEEE Frontiers in Education Conference (FIE), Rapid City, SO. 12-15 October.