Debugging performance: creative solutions to developing computer science students' problem-solving skills

Paul Miller
RMIT University, Melbourne, Australia
paul.miller@rmit.edu.au

Meg Colasante
RMIT University, Melbourne, Australia
meg.colasante@rmit.edu.au

Xiaodong Li
RMIT University, Melbourne, Australia
xiaodong.li@rmit.edu.au

Programming is an area of learning in computer science that students struggle with, in particular, they debug software poorly. Debugging is a key problem-solving skill in computer science and IT industries. This study examined how novices, undergraduate programming students, go about their debugging and how teachers guiding them into professional life can encourage improved performance. Using action research methodology, a new prototype debugging assessment strategy was developed, evaluated, and advanced. This paper focusses on an action research phase where qualitative data collection methods involved artefact analysis of online activities of students undertaking debugging assessments under computer laboratory trial conditions,; and both student and teacher focus group interviews. Themes emergent from the data analysis included relevance and transferability of debugging skills, and improvements identified for the strategy. This study resulted in benefits toward addressing key gaps in student learning, and reinforces the value of authentic learning guidelines.

Keywords: authentic assessment; debugging; problem-solving skills

Introduction

Skills required in computer programming are difficult for students to accomplish, which negatively impacts the student experience and their preparation for employment. In particular, students have poor practices in debugging software. This problem-solving skill involves methodically seeking out and reducing defects in a computer program code (i.e. in programming language statements). The computer science accrediting body, Australian Computing Society (ACS), describes necessary employment skills such as "testing, debugging, diagnosing and correcting errors and faults in an applications programming language within established testing protocols, guidelines and quality standards to ensure programs and applications perform to specification" (ACS, 2013, p.19). Likewise, debugging software is a major cost for industry occupying 25-50% of project development time (Chuntao, 2008).

The aims of this study were to examine how novices go about their debugging tasks and how teachers guiding them into professional life can encourage improved performance. An object of the research was to improve recently deployed prototype assessed lab tasks, designed to address the skills gap, through broad consultation and seeking improved practices with key stakeholders, such as students and staff previously involved in learning/assessing debugging skills. An initial teacher-initiated action research cycle resulted in the creation of a new laboratory assessment for debugging that was authentically aligned to industry practice. From this, another two cycles advanced and evaluated the strategy for teaching and assessing debugging skills. This paper reports specifically on the middle action research phase (of three phases) that evaluated the previous implementation cycle and tested new ideas with volunteer computer science students, who completed the course (i.e. semester subject) Advanced Programming Techniques in the previous year, and the course teaching team. Further reflections fed forward into a third cycle (planning and implementation in classroom conditions of an improved model, currently under analysis).

This paper offers a creative solution to an issue relevant to many computer science teachers across the sector. It also offers advice for a scholarly approach to designing assessment for quality learning, where the student experience is aligned to an authentic professional-in-the-making approach to better prepare them for graduate careers.

Literature review

Research shows that debugging is something that students do poorly (Chen, Wu & Lin, 2013; Chiu & Huang, 2015; Radermacher, Walia & Knudson, 2014). Beginner programmers have great difficulty debugging even their own code, let alone code created by others. They particularly have trouble finding the location of a bug and very few of them use formal tools to shortcut this process. There is also a large gap between debugging behaviours of novices and experts where these very shortcomings in novices are the things that expert programmers do well. Several studies conducted across educational sectors illustrate these issues.

One vocational high school study surveyed 72 twelfth-grade students after they completed two one-year programming subjects (Chen et al., 2013). They found that over half viewed themselves as lacking debugging skills and thought that such skills were accumulated with experience rather than learnt. In another senior high school study, 41 students participated in a six-week experiment on debugging skills. The students were given guidance worksheets on how to apply strategies to find programming errors and correct them. However, while a majority indicated the worksheets helped guide them to apply debugging strategies, more objective pre- and post-test performance measurements demonstrated these skills were not enhanced (Chiu & Huang, 2015).

In the higher education context, one project noted a gap between industry expectations and graduating computer science and software engineering student skills (Radermacher et al, 2014). It cited a case where four out of eight new employees struggled to effectively use debugging (diagnostic) tools to fix errors in their programming code (Begel & Simon, 2008, in ibid.). For their own study, Radermacher and colleagues (2014) interviewed 23 software industry employers to identify skill areas where recent graduates struggle. Problem solving and proficiency with debugging tools featured prominently in the responses. The authors suggested “knowledge deficiencies such as problem solving may require a more careful examination of the curriculum to determine if students are receiving adequate levels of instruction… [although some] knowledge deficiencies such as job expectations may be

difficult to incorporate into… course work" and offered that project-based capstone courses or industry internships may provide the best solutions (p.299). However, capstones and internships typically occur in the culminating stages of a learning program, which can be 'too little, too late'. Debugging skills need be scaffolded across a range of domain-specific learning subjects to allow simultaneous maturation of problem-solving and programming skills. Sheard and colleagues (2014) note a trend towards problem solving for first year programming students and acknowledge this "needs investigation to determine how students respond to learning programming in these environments" (p.25).

It has been long known in computer science education that quality learning experiences

> enrich students with increased knowledge and skills... [and motivate] them to solve challenging problems and fills them with the thrill of accomplishment... [resulting in] a clearer sense of who they are, and are ready to take greater responsibility for their education (Shneiderman, 1998, p.25).

However, creating such an engaging learning environment is not easy (Shneiderman, 1998). Additionally, it is harder to create quality assessments involving mastery compared to those that test recall of isolated facts (Ramsden, 2003). Two-decades ago Boud (1995) warned that assessment practices in higher education that have greater emphasis on summative judgement than quality learning could undermine learning. Boud questioned relationships between what is assessed and what students are expected to do after graduation, indicating a need for authentic assessments. Later, Boud with colleague Falchikov (2006), reiterated that higher education traditionally prepares students for acquisition over participation in learning, offering decontextualized learning isolated from practices or challenges of practice. While crediting improvements to assessment, Boud and Falchikov (2006) claimed a divide between learning experiences of a student compared to learning as a practitioner.

In preparing graduates for industry roles, students need skills and knowledge that can be authentically applied in industry. Herrington and Oliver (2000) established nine characteristics/elements to promote authentic learning experiences, with an update by Herrington, Reeves and Oliver (2014) offered here paraphrased (and heavily simplified).

- authentic contexts: complex, purposeful; motivates exploratory learning; reflects application in real-life
- authentic activities: ill-defined requiring subtasks; applicable across various subjects; seamlessly integrated with assessment
- experts and modelling: access to expert thinking/performance/modelling
- multiple perspectives: exposure to and exploration from other points of view
- collaborative construction: collaboratively construct knowledge and solve real-world problems
- reflection: meaningful reflection on learning; comparison opportunities with others
- articulation: express understanding, reasoning, weaknesses or gaps in thinking/learning; opportunities for articulation of ideas

- coaching and scaffolding: some deliberate facilitation/coaching and scaffolding supports

- authentic assessment: tied to successful solution of task; demonstration of effective performance with new knowledge.

These characteristics of authentic learning form design guidelines, not binding elements, that enable gauging of a learning situations' authenticity as if across a continuum (Herrington et al., 2014). They also provide a useful checklist to gauge authenticity of learning and assessment activities post hoc (e.g. Colasante, 2011).

Methodology

Action research, the methodological approach used in this inquiry, enabled cyclical practice development and theory to be tested during practice (Noffke & Somekh, 2011). "[A]n impetus for change/innovation" was the driving force (ibid. p.97) as attested by the aim of this project. That is, in order to better understand and work toward solving the issue of why novices perform so poorly at problem solving 'debugging' tasks, the aim of this research was to examine how students do so, and how teachers guiding them into professional life and employment can encourage them to improve performance. The pragmatic and democratic nature of action research allowed the researchers and participants to take a shared approach in deliberations and actions toward developmental outcomes (Levin & Greenwood, 2011).

While each research phase involved the widely accepted processes of planning, acting, observing and reflecting (Hine, 2013), the element of action remained central by "acting in a real context, reflecting on the results, and then acting again" (Levin & Greenwood, 2011, p.29). Specifically, actions involved "interventions in the social situation" (Noffke & Somekh, 2011, p.97) and are described in Table 1.. Note that while three major action research phases were undertaken, this paper primarily focuses on the middle phase, which included reflecting back onto the first phase, and planning forward for the third phase.

Table 1: The three macro research phases (focus on middle phase)

Phase	Timing	Action
1	2014, Semester 1 & 2	Acted on observations, and in consultation with others, developed and piloted a prototype assessment strategy over two micro cycles (two cohorts) aimed at improving students' debugging skills. Applied for competitive internal grant to support further cycles.
2	2015, Semester 1	Evaluated pilot implementation; planned next actions. Further developed assessment strategy, tested and observed under trial conditions; reflected on strategy in collaboration with key participants to plan next implementation.
3	2015, Semester 2 (under analysis)	Implemented further developed assessment strategy in full classroom conditions, reflected on strategy in collaboration with key participants.

The first cycle began as routine scholarly practice. In the processes of subject redesign, the head teacher/lead researcher utilised his observations of repeated poor performance in debugging to stimulate discussions with colleagues. From this, he designed a new assessment

and learning strategy around debugging a program written by another programmer, reflecting real-world debugging activities. This prototype assessment-for-learning exercise intended to illustrate the important component skills required to perform debugging via an experiential approach. Initial success in implementation gave encouragement to proceed and indicators of what to do next (e.g. a common debugging assessment, that is, the same faulty programming code for all students, may have allowed communication opportunities between scheduled laboratory groups and potential for unfair advantage).

The middle cycle marked commencement of the funded project and data collection. It involved evaluating the work of the previous cycle of implementation, further improvement of the assessment strategy, and testing with volunteer students. After the pilot round, major improvements in the strategy were the design of a class set of real-world debugging exercises—so students were not all repairing the same code—and the requirement for students to formally articulate their debugging activities.

Cohort descriptor

Advanced Programming Techniques (APT) is a first year core subject in the Bachelor of Computer Science degree but is also available as an elective subject. It typically has between 120 to 200 undergraduate students enrolled per semester. In the 2014 classes, which were the recipients of the pilot debugging assessment strategy, there were 158 (Semester 1) (of which 20 were female) and 218 (Semester 2) (of which 29 were female) undergraduate students. There is a trend of majority male students with 2014 no exception. For example, Semester 2, 2015 had 228 enrolments of which 18 were female.

Data collection

Data collected involved qualitative methods to harness both student and teacher reflections on their debugging learning and teaching activities from 2014, and their perceptions of the new instruments. Additionally, and prior to their group interview, volunteer students tested a few of the newly designed debugging assessment instruments under computer laboratory trial conditions. That is, students who had studied APT in 2014, and therefore experienced one of the prototypes, were invited via email to trial the advanced debugging assessment instruments. Four previous students volunteered and participated in the one-hour lab activity followed by the one-hour group interview, each receiving a token of appreciation ($25 gift voucher). The resultant student data collection involved

- Artefact analysis of their ‘process articulation’ form, where students completed a predominantly open-ended response Google Form (online Google survey tool) as they progressed through the trial, entering details about steps of their debugging processes.

- Focus group interview: immediately following the trial, based on five key semi-structured interview questions (audio-recorded and transcribed).

The 2014 teaching team (predominantly casual tutors) were invited to participate in a focus group interview of approximately one-hour. The first part of their interview involved moderation and validation of the assessment instruments, followed by more traditional semi-structured questioning. Each participating teacher also received a token of appreciation ($25 gift voucher). A research assistant facilitated the session, where the principal researcher was also a participant (audio-recorded and transcribed).

The focus group question sets comprised four to five key open-response questions, some of which had trigger questions in readiness for further probing. Such a semi-structured approach is appropriate "to diagnose potential problems with a new program, product, or service" (Stewart & Shamdasani, 1990, p.74), or to harness feedback on new assessment instruments as in the case of this project. Key focus group questions for the student participants related to levels of difficulty/clarity, and advice for future students undertaking such an assessment. Importantly for this paper, they were also asked whether they saw the relevance of this assessment for their ongoing learning and skills needed for employment, and suggestions on how to make the assessment better. Key focus group questions asked of the teaching team related to effectiveness of the debugging assessment, and other ways to scaffold students' learning. Additionally—and also important for this paper—they were asked whether this was the right kind of assessment for debugging skills, what alternate strategies might work better, and whether debugging should be emphasised as a skill requirement.

University ethics clearance was gained prior to each data collection cycle. All data collection processes were administered by research assistants not directly involved with the subject.

Data analysis

The data from the articulation artefacts (students' descriptions of their debugging processes) and the transcribed student and teacher focus group interviews were analysed. Two of the researchers independently immersed themselves in the data to draw out emergent codes, and then compared codes for similarities and differences until consensus was reached. They used their familiarity with the data and emergent coding to further code the data selectively to theoretical concepts, including authentic learning, and processed the data in NVivo software. Relevant theory can influence data analysis processes in action research, as can the overall aim of the research (Noffke & Somekh, 2011), resulting in several sets of interwoven codes, emergent and theory-generated. However, the emphasis of validating outcomes from action research rest in "testing out of conclusions through new actions, to see if the expected improvements result" (ibid. p.97), thus the data analysis particularly sought out such elements, looking for patterns and outliers, to determine effectiveness or otherwise of assessment instruments, and identify further areas for improvement.

Findings

The findings represent a slice of the second action research phase. In particular, analysis of the students' articulated debugging processes and achievements (artefact analysis) and the student and teacher focus group interviews helped to answer queries on relevance of the assessment strategy and what improvements could be identified. This rose from two emergent themes identified: transferability of debugging skills; and potential improvements or alternative learning and assessment strategies.
Note: instances of multiple uses of extraneous filler words ("like"; "kind of") have been removed from quotations for ease of reading.

Debugging achievements

The students' descriptions of their debugging processes in the trial revealed a number of insights. See Table 2 for a tabulated summary of artefact analysis of the qualitative 'process articulation' survey.

Table 2: Quantitative overview and notes of students' articulation of their debugging assessments

Student	Number of bugs identified	Ratio of successful:unsuccessful debugging	Observations on process
A	3	2:1	The two solved bugs were only 'reasonably' solved, i.e. a little clumsily.
B	3	3:0	One bug was clumsily corrected and insufficiently described. Another two were well corrected and articulated, including one at a high level of expertise.
C	3 (+ 1 duplicate + 1 non-bug)	3:1	Reported five bugs, however, one was a duplicate (albeit this was recognised), another not a bug (transitioned valid code to valid code). Attempts to correct the non-bug demonstrated a novice mistake and damaged the code. Two were corrected with sound methodological approach, and one a little clumsily via guesswork.
D	5	3:2	Sound methodology was used to solve 3 bugs, but the 2 not correctly solved showed gaps in knowledge and skill, including transitioning valid code to invalid code.

The students A and B, and arguably C, were not able to find more than three bugs in the code provided in their trial assessments. This may be reflective of poor understanding of program structure (to be further explored); however, it may also indicate that an imposed time period was restrictive to completing the task. Students B, C and D provided examples of mixed skill levels. They demonstrated sound debugging skills in some cases, and made obvious errors in others. Student C demonstrated reliance on guesswork in at least one debugging attempt. Student D did not utilise the full features of a debugging tool. Not understanding (i.e. being able to correctly interpret) or not utilising available features results in time wastage. Such errors are symptomatic of a novice programmer.

It may be argued that debugging is a more sophisticated skill than debugging and should be delayed in being taught but that has created problems and barriers to the development of our students that this project was intended to correct. For example, Bryce et al (2010) argue "One cause for attrition is that many students become frustrated with programming bugs" (p.6).

Relevance and transferability of debugging skills

In the teaching team interview, all teachers reinforced the importance and need for debugging in the learning of computer programming. For example

> It's a bit of a learning curve. But they have to do it [debugging]. Otherwise…you can write programs but that's your program. In the industry you won't get your program. You have to deal with someone else's program. You have to understand it. (Teacher)

This was in response to a question asking if debugging should not be highlighted as an important skill to learn created discussion. It prompted the trigger question 'Should this be a skill which is generalised throughout the curriculum?' This sub-question received unanimous support with the proviso that more time was required. The teachers emphasised the need for students to transition through novice skill level toward expertise. They discussed novices who used guesswork or asked others, as compared to those progressing along an appropriate learning trajectory. One teacher summarised the varying abilities on saying

> [Some students] had an idea of debugging in general, perhaps, or were capable enough of thinking logically and stepwise, "if this is going wrong where I should start", and connecting that to what they have learned in terms of the programming language… compared to those who just randomly "oh, this is not working, where is this line, why?" and just not even trying to relate it back to where it came from. (Teacher)

Comments from the student focus group on whether they saw relevance for debugging for ongoing learning and skills required for employment, indicated that they valued the opportunity and saw a need for it into their future. For example

> Learning how to use valgrind and GDB [diagnostic tools] probably saved my butt last semester when I did… [another computer science subject]. It helped a lot whenever I was having like segmentation errors or when something wasn't copying properly. (Student)

> ...debugging is always going to be a part of programming if that's your career choice, and having those skills now... [helps toward] experience; the more experience you have doing it the better. (Student)

The students identified transferability difficulties with debugging relating to reorientation to both the tools and processes. For example

> …first time seeing them [the source files to debug] and just remembering where the functions are. (Student)

> I'd... forgotten my GDB [diagnostic tool] commands... [which] made my workflow run a lot slow [sic.]... Once I'd got over that it was... just kind of basic problem solving. (Student)

On one hand the need for reorientation leans toward a 'use it or lose it' maxim, in that the year following their participation in the new debugging strategy they needed to spend some time reorienting. On the other hand it also suggests that the students were able to successfully self-orient to debugging tools and processes despite several months interlude and apply them to an unfamiliar piece of code (albeit the artefact analysis showed a mixed level of practice).

Improvements or alternative strategies

In the teacher group interview, there was lengthy discussion of a pedagogical nature, related to learning needs, curriculum constraints, and within these contexts how to enhance the quality of learning and assessment of debugging skills. An idea that emerged but not settled unanimously was segmenting the tasks into mini-tests or tutorial questions, such as "what is wrong with this line of code?" (Teacher). Another idea of offering an example exercises emerged from both teacher and student groups to "get used to the commands" (Student) and familiarise with "the type of program... [students] are going to need to debug" (Teacher).

However, there was one idea that emerged and solidified over the course of the assessment validation and moderation, through the teacher group interview, and reinforced at the end of the session. That was time allocation to task. An excerpt of the final conversation (abridged):

> Okay, there is definitely one common thing that has been raised... If we give them more time they will be able to figure out more, be a bit more relaxed. Given this is their first… time debugging someone else's code, I suppose, so maybe time pressure is not such a good thing. I believe most of them will benefit if they had more time. But it's still important. The level of difficulty still needs to remain the same, just give them more time... it's a more realistic scenario... . (Teacher)
>
> The more relaxed they can do a bit of search on various search engines and they can perhaps try to figure out… maybe use the tools better. (Teacher)
>
> They can experiment a lot more with more time. (Teacher)

The student group also raised time as an issue, for example, one student reflected on the 2014 experience saying:

> I feel like as a student I just haven't had time to get used to using [diagnostic tools] GDB and valgrind by the time the debugging lab comes across… an hour is the longest time you've ever spent just kind of figuring it out. (Student)

A student conversation centred on the challenge of using advanced editing tools, thus some reversion to more comfortable tools to apply to simple debugging processes. However, they then found they weren't adequately prepared to use the advanced tools when they really needed to. This suggests a need for further scaffolding of the students learning and perhaps further explanation of learning purpose to increase their motivation to use the more challenging diagnostic tools.

Discussion

The debugging assessment strategy aligned with a number of the nine characteristics of authentic learning (Herrington et al., 2014), although some improvements can be extrapolated. The strategy provided an authentic context applicable to eventual workplace practice, within the confines of limited laboratory time creating a one-hour restriction. The students in the trial found and corrected a limited number of bugs, indicative of both skill level and/or insufficient time to complete the exercise. A one-hour time period is more reflective of university timetabling than of authentic industry practice, or indeed, space needed for learning. The shorter time may have contributed to higher cognitive load and introduction of errors which may not have occurred otherwise. The teachers collectively determined to solve this by extending the time available into non-timetabled time, to a full day exercise. They could have instead chosen to reduce the complexity of the assessment. This would have rendered the activity less realistic and less useful.

The authenticity of the debugging activities was noted in the transferability across other programming subjects, and projected usefulness of workplace application. Additionally, the students in the trial session were tasked with articulating their debugging processes into a Google form. While this prompted them to express and reflect on the steps they took and their successes/non-successes, it also allowed the researchers an additional view into their knowledge and skills. "There can be no truthful reporting or effective changes to teaching in the absence of faithful diagnosis of students' understandings" (Ramsden, 2003, p.205). This articulation part of the assessment was initially proposed as both a data collection tool and a trial grading method but was determined to be a useful innovation that in the future should

aid teaching staff to determine student understanding. It also actually helped to ensure authenticity of the assessment by enabling explicit demonstration of effective performance with new knowledge (Herrington et al, 2014). Indeed, this articulation component was implemented into classroom practice in the third phase of the action research.

A couple of authentic learning elements need further consideration. The first is the indicator for more coaching/scaffolding for the students to help them transition from 'comfortable' editing tools to the challenges of using more complex, and subsequently more useful, tools. Additionally, the findings revealed (unsurprisingly) that debugging skill levels vary between students. However, the artefact analysis demonstrated that these may vary for individual students from one problem to another. This issue of transition from novice to expert requires further analysis. The researchers have earmarked a need to analyse indicators to identify when students are still within a 'zone of proximal development' (Vygotsky, 1978) with the aim to guide them toward mastery using appropriately aligned supports.

Another element that raises interesting questions for further analysis is that of collaborative construction, particularly collaborative construction versus plagiarism. However, this goes beyond the axiomatic use of such phrases that can bind teachers to traditional, more simplistic assessment measures. It is based on finding the right balance between real-world workplace collaboration compared to students sharing debugging solutions between laboratory sessions. Large class numbers render multiple lab sessions necessary, and the design and implementation of multiple different pieces of faulty code to debug was the trialled solution. This was explicitly implemented in the third phase of the action research.

Conclusion and further work

This action research study significantly addressed key gaps in student learning, and reinforced the relevance and transferability of debugging skills across learning contexts and into the workplace. Such skills need to be aligned to authentic conditions of workplace practices. A subsequent action research phase has followed, responding to the examination and reflections from the findings reported in this paper. This next cycle examined the further evolved assessment strategy in a real classroom context, for a new cohort of semester 2, 2015 students. Data collection involved mixed methods, based on the qualitative methods reported here plus student recordings their computer screens as they underwent the assessment activities and articulation of their debugging processes, and student assessment results and course experience survey data. Examination of the evolved strategy in real practice was postponed to ensure the analysis of the previous research cycles were complete, allowing quality improvements and readiness of the strategy to 'go live'. The data from the latest round is currently under analysis to be reported elsewhere.

Acknowledgements

This project received contested RMIT University funding in the form of a 2015 Scheme for Teaching and Learning Research (STeLR) grant, sponsored by the College of Science, Engineering and Health Academic Development Group.

References

ACS. (2013). *ANZSCO Descriptions – V1– 2013*. Australian Computing Society, retrieved https://www.acs.org.au/__data/assets/pdf_file/0018/7641/ANZSCO-Descriptions.pdf

Boud, D. (1995). Assessment and learning: contradictory or complementary? In P. Knight (Ed.) *Assessment for Learning in Higher Education*. London: Kogan Page, 35-48.

Boud, D. & Falchikov, N. (2006). Aligning assessment with long-term learning. *Assessment & Evaluation in Higher Education*, 31(4), 399-413. doi:10.1080/02602930600679050

Bryce, R.C., Cooley, A., Hansen, A. & Hayrapetyan, N. (2010). A one year empirical study of student programming bugs. In Proceedings of ASEE/IEEE Frontiers in Education Conference, 1-7. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5673143

Chen, M-W, Wu, C-C & Lin, Y-T. (2013). Novices' debugging behaviors In *VB programming. Learning and Teaching in Computing and Engineering (LaTiCE)* (pp.25–30). doi: 10.1109/LaTiCE.2013.38

Chiu, C-F, & Huang, H-Y. (2015). Guided debugging practices of game based programming for novice programmers. *International Journal of Information and Education Technology*, 5(5), 343-347. doi: 10.7763/IJIET.2015.V5.527

Chuntao, D. (2009). Empirical study on college students' debugging abilities in computer programming. *Proceedings of the 1st International Conference on Information Science and Engineering (ICISE2009)* (pp.3319-3322). Nanjing, China: IEEE. doi: 10.1109/ICISE.2009.550

Colasante, M. (2011). Using a video annotation tool for authentic learning: A case study. *In Proceedings of Global Learn Asia Pacific 2011* (pp. 981-988). AACE.

Hine, G.S.C. (2013). The importance of action research in teacher education programs. In Special issue: Teaching and learning in higher education: *Western Australia's TL Forum. Issues in Educational Research*, 23(2), 151-163. http://www.iier.org.au/iier23/hine.html

Herrington, J., & Oliver, R. (2000). An instructional design framework for authentic learning environments. *Educational Technology Research and Development*, 48(3), 23-48.

Herrington, J., Reeves, T.C. & Oliver, R. (2014) Authentic learning environments. In J.M. Spector et al. (eds.), *Handbook of research on educational communications and technology*, pp.401-412, New York: Springer Science+Business Media. Doi: 10.1007/978-1-4614-3185-5_32

Levin, M. & Greenwood, D. (2011). Revitalizing universities by reinventing the social sciences. In N.K. Denzin & Y.S. Lincoln (Eds) *The SAGE Handbook of Qualitative Research* (4th ed.), (pp.27-42). Thousand Oaks, CA: SAGE Publications.

Noffke, S & Somekh, B. (2011). 'Action research'. In B. Somekh & C. Lewin (Eds) *Research methods in the social sciences* (pp.94-101). London, UK: SAGE Publications.

Radermacher, A, Walia, G. & Knudson, D. (2014). Investigating the skill gap between graduating students and industry expectations. In *Companion Proceedings of the 36th International Conference on Software Engineering (ICSE Companion 2014)*, New York, NY: ACM, (pp. 291-300). doi: 10.1145/2591062.2591159

Sheard, J., Morgan, M., Butler, M., Falkner, K., Weerasinghe, A., Simon. (2014) *Experiences of first-year students in ICT courses: Good teaching practices*. Australian Council of Deans of ICT (ACDICT) Learning & Teaching Academy (ALTA) (Commissioned Good Practice Reports for 2013–2014).

Shneiderman, B. (1998). Relate-Create-Donate: A teaching/learning philosophy for the cyber-generation. *Computers & Education*, 31(1), 25-39. doi:10.1016/S0360-1315(98)00014-1

Stewart & Shamdasani, (1990). *Focus groups; Theory and practice*. Newbury Park, CA: SAGE Publications.

Vygotsky, L.S. (1978). *Mind in society; The development of higher psychological processes*. Translated works 1930s-1960s compiled by M. Cole, V. John-Steiner, S. Scribner & E. Souberman (Eds.). Cambridge, MA: Harvard University Press.