Metamorphic Testing:
A New Student Engagement Approach for a New Software Testing Paradigm

Dave Towey*
School of Computer Science
The University of Nottingham Ningbo China
Ningbo 315100,
People's Republic of China
Dave.Towey@nottingham.edu.cn

Huai Liu
Australia-India Research Centre for Automation Software Engineering
RMIT University
Melbourne VIC 3001, Australia
huai.liu@rmit.edu.au

Tsong Yueh Chen, Fei-Ching Kuo
Department of Computer Science and Software Engineering
Swinburne University of Technology
Victoria 3122, Australia
{tychen, dkuo}@swin.edu.au

Zhi Quan Zhou
School of Computing and Information Technology
University of Wollongong
Wollongong NSW 2522, Australia
zhiquan@uow.edu.au

***Abstract*—Software testing, as part of the software quality assurance processes in software engineering (SE), is seldom cited as the most engaging part of a software engineer's training. In spite of its importance, it was often a neglected area, and only recently, with the adoption of more agile methodologies, has testing come to the fore. Testers face challenges, including the need to be able to identify when software is not behaving as expected. Metamorphic Testing (MT) first appeared twenty years ago in an effort to help testers facing the oracle problem (not having a mechanism to determine correct software behavior). Although MT is simple in concept, it has demonstrated very impressive fault-finding ability, and has recently been attracting a great deal of interest. Training students and practitioners to effectively apply MT has become an important goal, but to date, no textbook or guidelines have been made readily available. This paper, written by some of the leading voices in the MT community, is a sharing of some of the experiences, reflections and insights gained through teaching MT (and other related subjects). These experiences indicate that not only is MT effective in industry and at fault-finding, it has the potential to really engage students in creative SE classroom activities.**

Keywords—Teaching Software Engineering; Teaching Computer Science; Metamorphic Testing; Autoethnography; Reflection.

I. INTRODUCTION

Changes in education around the world have led to innovations in the classroom [1]. New technologies and techniques have been delivering more resources to both teachers and students, and pushing the boundaries of teaching and learning experiences [1]. It is important to note, however, that not all disciplines have embraced the opportunities offered by these innovations and new technologies, and ironically, computer science (CS) subjects — arguably the source of many technical advances — have not always been amongst the first adopters [2], [3]. Armarego [4] reported that creativity and multidisciplinary skill development are missing in traditional software engineering (SE) curricula, suggesting a need for more measureable high-level intended learning outcomes, and inclusion of apprenticeship, problem-based learning (PBL) or studio-based learning in SE teaching and learning activities.

This paper presents some reflections on a number of classroom experiences delivering SE-related modules. It draws on the collective experience of five academics, working in different teaching contexts, all of whom share an expertise in the software quality assurance area of software testing, and particularly of Metamorphic Testing (MT) [5]–[8]. The MT experiences indicate that it is both a powerful tool in industry, and a new paradigm for considering, and teaching, software testing (ST).

The rest of this paper is laid out as follows. Section II introduces and describes some of the main themes and concepts, including MT, and SE education. Descriptions of some concrete SE teaching experiences are presented in Section III. Section IV contains the various reflections and insights based on the SE experiences. Some final conclusions are given in Section V.

II. BACKGROUND

In this section, we present some of the background to the paper, including introducing software engineering (SE), software testing (ST), and Metamorphic Testing (MT). We discuss some issues surrounding the instruction of SE and ST, and the need for reflective thinking in our teaching practice.

* Corresponding author

978-1-5090-5598-2/16/$31.00 ©2016 IEEE 7-9 December 2016, Dusit Thani Bangkok Hotel, Bangkok, Thailand
2016 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)

Finally, we end this section by giving a more detailed introduction of ourselves.

A. Software Engineering

Software Engineering (SE) can be defined as "the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software" [9]. SE is often traced back to a NATO conference in 1968, where the pairing of the words "software" with "engineering" was a deliberate call to practitioners to apply structured, rigorous "engineering" approaches to address the then software crisis [10] — it was a time when software development ran over schedule, over budget, and often ended up producing low quality systems ... or not producing a system at all [11].

The cause of this crisis was famously described in Dijkstra's 1972 Turing Award lecture [12] in terms of computing power outpacing human ability to harness it:

> "The major cause of the software crisis is that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem." [12, p.861]

Bringing engineering methodology to bear on the problem of software development was, as Dijkstra again said, expected to make it possible to "design and implement the kind of systems that [were] straining our programming ability at the expense of only a few percent in man-years of what they cost us [then], and [...] these systems [would] be virtually free of bugs" [12, p.863].

Although Dijkstra's vision was not realized — and some might argue that the crisis persists even now [11] — over its half-century history, SE has evolved and grown significantly. The most recent Software Engineering Body of Knowledge (SWEBOK) [13] now lists fifteen sub-disciplines, or key knowledge areas (KAs), spanning the entire SE lifecycle, from requirements gathering to software testing, evolution and maintenance.

B. Software Testing

The KA most relevant to this paper is software testing (ST), which SWEBOK [13, p.4-1] describes as consisting of "the dynamic verification that a program provides expected behaviors on a finite set of test cases, suitably selected from the usually infinite execution domain." In other words, ST involves execution of some software with a goal of uncovering faults. Each execution is achieved by applying the combination of inputs necessary for a single use of the software — called a test case [9]. Except for the most trivial of programs, execution of all possible test cases — *exhaustive testing* — is not usually possible, and hence some selection strategy is necessary. Ideally, the selected test cases would reveal all faults in the software; and successful execution without revealing faults would imply program correctness. This challenge of choosing an appropriate set of test cases is referred to as the reliable test set problem [14].

Related to the reliable test set is the question of how to know, when testing, if the software is providing correct behavior and output — and how to know when there is a fault. A smoke test is a most basic test, which reveals problems by actually obviously breaking the software — in days of hardware testing, this would result in smoke emitting from the machine. Beyond smoke tests, though, the ability to categorically determine the correctness of software behavior, and especially output, is another significant challenge to software testers: If a mechanism exists to make the determination, then it is said that an *oracle* exists; if there is no such mechanism, or if it is not practical to use the mechanism, then the testers face the oracle problem [15].

C. Metamorphic Testing

Many of the strategies and approaches that have been developed to address the reliable test set issue [14] may not be applicable in testing situations facing the oracle problem [15], which can significantly reduce a tester's ability to conduct effective testing. Metamorphic Testing (MT) was first proposed in the context of alleviating the oracle problem [16], and has developed a steadily increasing popularity, both in academia and in industry [5]–[8]. MT's simple, but effective, approach has successfully uncovered software problems, even in extensively tested systems [17].

A key aspect of MT is the set of Metamorphic Relations (MRs), which are identified based on properties of the program to be tested. Each MR represents a relationship among several inputs and their corresponding (expected) outputs — instead of trying to confirm the correctness of each individual output, MT checks the program against selected MRs. Testing with MT involves two or more executions of the program under test, with input for the first execution(s) called the *source test case(s)*, and input for the related follow-up execution(s) called the *follow-up test case(s)*. If the output for source and follow-up executions violate an MR, then the program must contain a fault.

To illustrate the basic idea behind MT, consider the shortest path problem [18]: Given a program P that calculates the shortest path between any two vertices a and b in an undirected graph G (written as $P(G, a, b)$), then P should output an ordered list of points in G that form a path from a to b, such that no other path between a and b is of shorter length. If G is large, it can become very difficult to determine whether or not the output really is the shortest path between a and b: In other words, the tester of this program faces the oracle

problem. Using MT, the tester may identify some MRs, such as the following:

- $MR1$: If the vertices a and b are swapped, the length of the shortest path should remain unchanged, that is, $|P(G, b, a)| = |P(G, a, b)|$.
- $MR2$: If x is any point on the path output when the program is called with vertices a and b (i.e., the program gives a shortest path $< a, ..., x, ..., b >$), then $|P(G, a, x)| + |P(G, x, b)| = |P(G, a, b)|$.

To test the program, with $MR1$ for example, the tester would execute the program twice — once with the *source test case* (G, a, b); and once with the *follow-up test case* (G, b, a) — and verify the results by checking if the MR ($|P(G, b, a)| = |P(G, a, b)|$) is violated. If a violation is identified, then the program is faulty.

MT has been applied in many industrial contexts, and, in spite of its apparent simplicity, has been remarkably powerful in finding faults, even in programs that had already undergone extensive testing [17]. In addition to its industrial success, it has also been gaining popularity amongst educators [19]–[21]. The increasing number of academics working in MT will likely result in greater prominence of MT in SE and ST curricula.

D. Teaching Software Engineering

Becoming proficient in some aspects of software engineering may not necessitate a full university degree, and the many new open educational resources (OERs) [22], such as MOOCs, mean that it is easier than ever to acquire many relevant skills [23]. Nonetheless, formal courses, such as those typically offered by higher education (HE) institutions (HEIs) remain the most common pathway to professional practice [2].

Herbert et al. [24] found that many CS graduates who later became software testers had incomplete skills, suggesting that the lacking skills may need to be "supplied by another organisation or a postgraduate degree" [p.34]. Ng et al. [25], examining the testing practices in Australia, noted that only about ten percent of testing training courses taken by people in industry were actually offered by HEIs, suggesting that this may indicate a lack of practical software testing ability taught at tertiary level. Armarego [4] also reported that creativity and multi-disciplinary skills development are missing in traditional SE curricula, and suggested inclusion of more apprenticeship, PBL, or studio-based learning in SE teaching and learning activities.

E. Reflective Thinking

Higher education (HE) has been evolving, and as different models for learning and teaching have been developed, HE teachers have been finding themselves under increasing pressure to modify their teaching styles to better match student expectations. The switching from the traditional "sage on the stage" role of teachers to more of a "guide on the side" [26] has meant that academics also need to be able to facilitate and encourage learning. As the philosophy underlying teaching has moved towards a more "student-centered" approach, feedback and input from students have been identified as an essential part of the quality assurance implemented by HEIs [27], [28].

Although sometimes controversial, student feedback is playing an increasingly important role in teachers' success in their careers. The need to be able to adjust and adapt to such feedback, and indeed, the need to grow as teachers, through reflective practice, are hallmarks of the modern HE teacher [29]. → prob. important

The reflections in this paper have been partly guided by the theories of Kolb [30] and Schön [31], and, in some cases, also by autoethnography [32], [33], which, as one of the authors has previously described [34], can be considered a rigorous approach and framework for reflection. Autoethnography is a type of research involving "self-observation and reflexive investigation in the context of ethnographic field work and writing" [33, p.43]. Duncan [35], discussing Peshkin [36] and Eisner [37], stresses that the subjectivity of the author is not only included, but *emphasized*; and that it is the researcher him/herself, as "connoisseur and instrument," who makes the study possible.

F. The Teaching Team

The reflections in this paper come from a team of five academics working in four different HEIs, across two different continents, and representing a range of experience and rank. All five of us (Dave, Fei-Ching, Tsong Yueh, Huai, and Zhi Quan) are currently involved in research and teaching related to MT (Section II-C), but we have also taught, and worked, in other contexts. The team includes one of the originators of MT, and four of the global top ten MT authors [5].

Our professional teaching experiences, in MT and in other SE areas, have provided us opportunities to learn and develop our abilities. By experiencing, sharing, and reflecting, we have been able to compile a number of insights and ideas that have, we believe, facilitated more interactive and engaging classes.

III. SE/ST Teaching Experiences

In this section, we present some of our experiences teaching SE and ST.

A. Failed Flipped SE Classroom

Motivated by a desire to improve the interaction in a software engineering fundamentals (SEF) module, Dave describes an attempt he made to design and implement a flipped classroom [38], [39], through what he called the *SEF Class Preparation Strategy (SEFCPS)* [34]. He divided the class into teams, with each student belonging to a *pair*, several pairs being a *team*, and the entire class being composed of two

groups of teams. He then scheduled, over the course of the semester, a series of activities to facilitate the teams and groups working together, and which would also free class time for more meaningful and engaging interactions.

For each SEFCPS topic, materials were made available three weeks before the class session for that topic. In the first week, one group (several teams) used the materials and posted questions in the virtual learning environment. These questions were to help guide the second group's engagement with the topic: In the second week, teams from the second group were to answer the questions. In the third week, the one ending with the actual class session, the original question posters from the first group gave feedback on the second group's answers. All questions, answers, and feedback were visible to the entire class. Through this process, the entire class would be familiar with the topic, and would have worked together as teams to get this familiarity. When the actual class session took place, it was going to be possible to have in-depth discussions and challenging tasks, including those inspired by Dawson's *dirty tricks* [40].

Ultimately, the SEFCPS initiative was not as successful as had been expected, and a number of possible reasons were identified, including that the lack of explicitly allocated marks for different aspects of the activities may have left students not considering it worth their while [41]. Another conclusion was that, although a teacher may think he knows the best way to do something, the students may not agree: The students may analyze the problem and choose an alternative, apparently more cost-effective, solution. The SEFCPS was attempted without initial student consultation or input, nor was the possibility considered that the students would identify and implement an alternative *method* to achieve the apparent SEFCPS objectives — the students effectively *hacked* the preparation strategy [34].

Fei-Ching has also discussed attempting a flipped classroom approach. Learning from Dave's mistakes, students were consulted before implementation, but again using pairs, and allowing each pair to choose topics, study them, and propose questions for classmates. She emphasized that the proposed questions could form the basis for the final exam ... but, like Dave, was ultimately disappointed by the bookish, theory-oriented nature of the student-generated questions, which were also often left so late as to be of little actual use for preparing the class. This also echoes Dave's note that, when he asked students to prepare questions and feedback to help guide classmates' engagement with the course materials, he had overlooked how difficult this is for professional educators to get right, never mind for students without pedagogical training [34].

B. Problem-based Learning Assignment

Fei-Ching describes using a problem-based learning (PBL) [42] approach over two years in a module on the process and automation of software testing. Students were given an assignment to verify software quality, and test some software of their own choice — the problem was loosely defined and solutions were not fixed.

It was found that, because test automation was a main theme of the module, students seemed to want to automate checking the test outputs. Even though MT may have been a better way to handle any type of software testing (with or without the oracle problem), the majority of students avoided choosing software with the oracle problem, or chose to manually construct an oracle for a small number of specifically designed test cases. Only two of the twelve teams over the two years derived MRs and applied MT. When reviewing the module, Fei-Ching suggested that the lack of explicit marks for the oracle problem and MRs identification, may have been a reason for the lower than expected number of MT attempts. This fits with Dave's conclusion that the lack of allocated marks for the SEFCPS [34] may have caused students to not view the activity as worthwhile.

C. Challenging Projects

Project work has also been used to help teach MT.

Zhi Quan designed challenging software testing projects for the final year SE subjects he taught, including asking students to apply MT to test web search engines. Search engines are difficult to test due to the presence of the oracle problem, but Zhi Quan observed that students were able to design some appropriate MRs, and became excited when violations were detected. However, although many violations did indicate something wrong in the search results, Zhi Quan noted that some students were focused more on the violations, than on the possibility that the cause was indeed a bug (because finding the cause of a violation can sometimes be difficult, especially for complex systems) — Zhi Quan describes this phenomenon as overlooking the need to justify and validate the MRs themselves, and says it is something he has seen in other projects and modules. He suggests, as elsewhere, that fundamentals, such as rigor in approach and design, need to be emphasized before attempting to train students to use MT.

IV. REFLECTIONS AND INSIGHTS

In this section we discuss some reflections based on the various experiences described in Section III.

A. Student Insight and Hacking Skill

As in the case of Dave's flipped classroom (Section III-A), an often overlooked aspect of teaching — especially in the more traditional, teacher-centered paradigms — is that the students may be able to contribute significantly to the learning activities and plans. The elaborate learning activities Dave attempted to implement were analyzed by the students, and rejected (without telling Dave) in favor of an alternative, student-developed strategy. Although disappointed by the

failure of his flipped class, he was nonetheless very impressed by the students' ingenuity. The ability of the CS students to "hack" a solution actually helped shape his later classes and engagement with those students — including in the very successful MR identification activity some time later (Section IV-C). copium... but academic

B. Incentives

Given the heavy workload that HE students often face [43], and external demands such as part-time jobs or social activities, it should not be surprising that they may make decisions based on maximum return for minimum effort — they apply cost-benefit analyses to their approaches to assignments. If it is clear how completion of homework in a certain manner may contribute directly to grades, it is easy for students to see the benefits — and vice versa (Sections III-A and III-B). If the objective is to get students to practice MT, it may be necessary to make it a compulsory task, and assign it marks [41]. someone got it!

Another perspective on this, again relating to student ingenuity (Section IV-A), may be that the students are very much aware of their strengths and weaknesses, and, naturally, would prefer not to expose weakness in graded assessments. Although a project may appear to lend itself to a particular style of solution, through certain programming techniques, for example, if the students have not yet sufficiently mastered those techniques, then the students may not use them in their solution. As with Fei-Ching's experience running PBL projects, although the students appeared enthusiastic about automating the verification process using MT — which required writing code — due to a lack of programming competence, many students seemed to avoid using MT.

C. MR Identification

Identification of valid MRs remains perhaps the single most challenging aspect for new MT users — it is certainly the most discussed topic in the MT literature [5]–[8]. It would seem obvious, therefore, that this is an area to be focused on in the classroom.

Dave has described a particularly satisfying classroom experience when, after introducing the basics of MT, the students discussed the properties and possible MRs for the Shortest Path program example (Section II-C). As with other student-centered learning activities, the students led the discussion, proposing different ideas as possible MRs. At one point, a student began describing several properties that Dave had not previously considered as MRs for this example. Although not unusual for trainees or students to identify MRs not previously thought of by the instructors or teachers (all of the authors have had such experiences [7]), this instance was particularly useful: It illustrated a perspective on the problem that later allowed Dave to explain a much more advanced MR identification example [44].

The perspective used by the student could be described as follows: If $P\ (G,\ a,\ b)$ outputs the path $< a,\ ...,\ b >$, then deletion of points not on this path should not impact on the length of the shortest path output by the program. After some further discussion and exploration of this concept, the class were able to formalize the MR. Dave then explained how Le et al. [44] had used a similar concept when examining C compilers: If the source code for two programs are equivalent for a particular input, then, after compilation, the corresponding executable programs should also be equivalent for the same input. Le et al. basically constructed two such programs by looking at which statements were executed in a given program for a given input, then removing some non-executed statements in that program to create the equivalent program (for that given input). Both programs were then compiled into executables, run with the given input, and any discrepancy in behavior was investigated as a possible fault.

This instance is further proof of the ability of students to exceed expectations in unanticipated ways (Section IV-A), and serves as a reminder to *guide from the side* [26]. This use of an appropriate example that students can readily understand is emphasized in MT teaching.

D. Appropriate Examples

Although it does not appear difficult for most students to grasp the basic concepts in MT, moving beyond this theoretical understanding to being able to actually apply MT in testing seems very challenging: Most students cannot use MT without appropriate training, involving appropriate examples. Such examples should start out relatively simple, and then progress to the more complicated: Start with things that most people can easily understand — like finding the maximum value in a series of integers — and then move onto the complicated examples, such as MT for compilers (Section IV-C).

Huai suggests using examples that fit with the learners' domain knowledge, but which are also *diverse*, spanning different domains, multi-disciplinary areas, languages, and functionality. Domain knowledge can be a critical factor in MR identification, and Huai has reported on training testers to identify MRs for the UNIX/Linux *grep* command: The trainees who knew the command were quickly able to identify appropriate MRs, but those who did not already know it, even though they had access to the technical specifications, were not able to identify any [7].

The most successful MT teaching involves learning by example and practice, using multiple examples, and progressing from simple to more complicated — as Dave's student made possible (Section IV-C).

E. MRs and Other Software Quality Assurance

Zhi Quan has found that when students first attempt MT, the properties they identify may not be good MRs, or may not even be MRs at all. A cause for this may be students

attempting to create MRs without closely following the software's specification or user manual. He suggests emphasizing the *formal* concept of "correctness" of programs [9] before attempting MR identification or other testing tasks.

Although challenging, the task of identifying MRs can also lead to other testing and validation. Huai reports on experiences with students and trainees not being able to identify valid MRs, but actually finding valid properties for other testing techniques, including assertion and invariant writing [45], and error-trapping for spreadsheets [46]. Zhi Quan has suggested that appropriately supported students may be able to think beyond MT and design newer testing approaches.

The opportunities to naturally expand learning beyond MT into other verification and validation are clear.

F. Topic Delivery

MT fundamentals can be taught in a comparatively short time [7]. In spite of a current lack of textbook, the simplicity of the MT basic concepts and methodology allow for its initial instruction to be done within a couple of hours — about the length of a single lecture period. Huai describes how, in his teaching, he stresses that MRs should not necessarily be limited to just two inputs — multiple source and follow-up inputs can be involved (Section II-C) — and reports that this concept seems readily accepted and grasped by students.

We have noticed how different approaches in the classroom have yielded different results. Huai discusses the trade-off between targeted training and open discussion, noting how open discussion can enable learners to show innovation in MR identification. We have all had sessions where students obtained novel and very effective MRs though open discussions (Section IV-C). On the other hand, some rigorous or specific instruction is also necessary to ensure the basic quality of the identified MRs, and that they can be valid, and nontrivial.

Although MR identification is the focus of much of the MT training and teaching, other aspects of MT also require attention. Some teachers may side-step the issue of where source test cases come from by using ad-hoc or even random test cases, however, our research experience has indicated that source test case generation is not so trivial that it can be ignored [7].

The student-centered and student-led approaches used to deliver MT-related topics do mean that many interesting and unexpected issues can arise in the classroom. Huai explains how he applies the agile techniques [47] taught in modern SE to his classroom management of MT topics, using an agile and iterative approach, adjusting to the student-led directions.

Tsong Yueh describes the best (MT) teaching approaches as being more apprenticeship-like in style, learning by example and practice — addressing Armarego's [4] call for SE curriculum reform.

G. Apprenticeship and Teamwork

The best way to improve at using MT, by getting better at identifying good MRs, is through practice, through *apprenticeship* with proficient and better MT users.

As with many other cases of teamwork [48], our experiences of teams in MT training have highlighted their strengths, even teams of only two people, which Tsong Yueh describes as performing better than two individuals in identifying MRs. He recommends using teams of up to about ten students, giving them an algorithm and some initial MRs, and helping them to identify more. As much as possible, the instructor can give feedback on the identified MRs, possibly guiding the teams towards better or more refined ones. Team members themselves can help each other correct and enhance identified MRs. Industry-related internships have also proven both popular and effective. Fei-Ching, for example, reports on how a supervised project from industry allowed students to explore MT in a real environment, identify MRs, and help the company find actual bugs.

H. Creativity

ST has been reportedly perceived as a menial, low-level job, lacking any need for creativity or ingenuity [49]. Armarego [4] has identified creativity as being missing from traditional SE curricula.

As reported in multiple studies (Section IV-C), MR identification is perhaps the most challenging aspect of MT for people just starting to use it. One reason for the difficulty is the need for creativity: MR derivation and definition require not only an understanding of software's requirements (that may involve multidisciplinary knowledge and skills), but also a creativity, and an ability to apply creativity in expansion of the MR.

Happily, as has been reported from many classroom and training experiences [7], [49], MT (and MRs especially), has prompted some of the most creative student work seen by the authors. Fei-Ching has argued that the creativity and problem-solving exercises in MT class exercises and projects help build the higher order cognitive skills reportedly lacking in CS/SE curricula [4].

V. CONCLUSION

This paper has presented some insights and reflections on teaching Metamorphic Testing (MT), an increasingly popular paradigm for software testing. MT was initially introduced to alleviate the oracle problem (situations where software testers may not be able to identify incorrect software behavior), and has been shown to be very effective in fault detection. The reflections were based on the teaching experiences of five leading MT researchers, who have found that, in addition to

the obvious fault-finding applicability, MT has enabled a kind of student engagement in the classroom that was not often seen before.

The reflections and guidance offered in this paper are certainly not exhaustive, but, we feel, may be sufficient to help teachers and trainers really incorporate MT in their own classrooms.

REFERENCES

[1] High Level Group on the Modernisation of Higher Education: Report to the European Commission on New Modes of Learning and Teaching in Higher Education. Luxembourg: Publications Office of the European Union, July 2014. [Online]. Available: http://ec.europa.eu/education/library/reports/modernisation-universities_en.pdf

[2] D. Towey, "Preparing the next generation of China's computer scientists: A snapshot of challenges for Sino-foreign computer science education," in *Proceedings of the Third International Conference on Open and Flexible Education (ICOFE 2016)*, Jul 2016, pp. 224–231.

[3] T. Wang, M. S. Y. Jong, and D. Towey, "Challenges to flipped classroom adoption in Hong Kong secondary schools: Overcoming the first- and second- order barriers to change," in *Proceedings of the 2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, Dec 2015, pp. 108–110.

[4] J. Armarego, "Constructive alignment in SE education: Aligning to what?" in Software Engineering: Effective Teaching and Learning Approaches and Practices. Hershey, PA: Information Science Reference - Imprint of: IGI Publishing, 2008, chapter 2, pp. 15–37.

[5] S. Segura, G. Fraser, A. B. Sánchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[6] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Fault-based testing without the need of oracles," *Information and Software Technology*, vol. 45, no. 1, pp. 1–9, 2003.

[7] H. Liu, F.-C. Kuo, D. Towey, and T. Y. Chen, "How effectively does metamorphic testing alleviate the oracle problem?" *IEEE Transactions on Software Engineering*, vol. 40, no. 1, pp. 4–22, Jan 2014.

[8] Z. Q. Zhou, S. Xiang, and T. Y. Chen, "Metamorphic testing for software quality assessment: A study of search engines," *IEEE Transactions on Software Engineering*, vol. 42, no. 3, pp. 264–284, 2016.

[9] ISO/IEC/IEEE, "Systems and software engineering – vocabulary," ISO/IEC/IEEE 24765:2010(E), pp. 1–418, Dec 2010.

[10] P. Naur and B. Randell, Eds., *Software Engineering: Report on a conference sponsored by the NATO Science Committee*, Garmisch, Germany, 7th to 11th October 1968. Scientific Affairs Division, NATO, 1969.

[11] R. N. Charette, "Why software fails [software failure]," *IEEE Spectrum*, vol. 42, no. 9, pp. 42–49, Sep. 2005.

[12] E. W. Dijkstra, "The Humble Programmer," *Communications of the ACM*, vol. 15, no. 10, pp. 859–866, 1972.

[13] IEEE Computer Society, P. Bourque, and R. E. Fairley, *Guide to the Software Engineering Body of Knowledge (SWEBOK(R))*, 3rd ed. Los Alamitos, CA, USA: IEEE Computer Society Press, 2014.

[14] W. Howden, "Reliability of the path analysis testing strategy," *IEEE Transactions on Software Engineering*, vol. 2, no. 3, pp. 208–215, 1976.

[15] E. T. Barr, M. Harman, P. McMminn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507– 525, 2015.

[16] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Department of Computer Science, Hong Kong University of Science and Technology, Tech. Rep. HKUSTCS98-01, 1998.

[17] T. Y. Chen, F.-C. Kuo, D. Towey, and Z. Q. Zhou, "A revisit of three studies related to random testing," *SCIENCE CHINA Information Sciences*, vol. 58, no. 5, pp. 052 104:1–052 104:9, Springer-Verlag, 2015.

[18] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numer. Math., vol. 1, no. 1, pp. 269–271, Dec. 1959.

[19] H. Liu, F.-C. Kuo, and T. Y. Chen, "Teaching an end-user testing methodology," in *Proceedings of the 23rd IEEE Conference on Software Engineering Education and Training*, IEEE, 2010, pp. 81– 88.

[20] K. S. Mishra and G. Kaiser, "Effectiveness of teaching metamorphic testing," Department of Computer Science, Columbia University, Tech. Rep. CUCS-020-12, 2012.

[21] K. S. Mishra, G. Kaiser, and S. K. Sheth, "Effectiveness of teaching metamorphic testing, part II," Department of Computer Science, Columbia University, Tech. Rep. CUCS-022-13, 2012.

[22] MERLOT, "MERLOT (multimedia education resource for learning and online teaching)." [Online]. Available: https: www.merlot.org/

[23] D. Towey, "Open and flexible learning as an alternative in mainland Chinese higher education," in *Emerging Modes and Approaches in Open and Flexible Education*, K. C. Li and K. S. Yuen, Eds. Hong Kong: The Open University of Hong Kong Press, 2014, chapter 2, pp. 12–16.

[24] N. Herbert, K. de Salas, I. Lewis, M. Cameron-Jones, W. Chinthammit, J. Dermoudy, L. Ellis, and M. Springer, "Identifying career outcomes as the first step in ICT curricula development," in *Proceedings of the Fifteenth Australasian Computing Education Conference - Volume 136*, ser. ACE '13. Darlinghurst, Australia: Australian Computer Society, Inc., 2013, pp. 31–40.

[25] S. P. Ng, T. Murnane, K. Reed, D. Grant, and T. Y. Chen, "A preliminary survey on software testing practices in Australia." in *Proceedings of the Australian Software Engineering Conference*, IEEE Computer Society, 2004, pp. 116–127.

[26] A. King, "From sage on the stage to guide on the side," *College Teaching*, vol. 41, no. 1, pp. 30–35, 1993.

[27] E. O'Grady, J. O'Reilly, J. P. Portelli, and C. Beal, "Putting the learner into the curriculum, not the curriculum into the learner: A case for negotiated integrated curriculum," *International Journal of Pedagogical Innovations*, vol. 2, no. 2, pp. 51–64, July 2014.

[28] C. Xie, D. Towey, and Y. Jing, "Current trends in the use of student input in teacher evaluation in universities in mainland China," in *Proceedings of the Fourth Asian Conference on Language Learning (ACLL2014)*, Osaka, Japan, April 17-20, IAFOR, 2014, pp. 12–21.

[29] J. Moon, *A Handbook of Reflective and Experiential Learning: Theory and Practice*. Taylor & Francis, 2013.

[30] D. A. Kolb, *Experiential Learning: Experience as the Source of Learning and Development*. Englewood Cliffs, NJ: Prentice Hall, 1984.

[31] D. A. Schön, Educating the reflective practitioner. San Francisco: Jossey-Bass, 1987.

[32] C. Ellis, *The Ethnographic I: A Methodological Novel about Autoethnography*, ser. Ethnographic alternatives book series. AltaMira Press, 2004.

[33] G. Maréchal, "Autoethnography," in *Encyclopedia of Case Study Research*. Thousand Oaks, CA: SAGE Publications, 2009, pp. 43–45.

[34] D. Towey, "Lessons from a failed flipped classroom: The hacked computer science teacher," in *Proceedings of the 2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE),*Dec 2015, pp. 11–15.

[35] M. Duncan, "Autoethnography: Critical appreciation of an emerging art," *International Journal of Qualitative Methods*, vol. 3, no. 4, pp. 28–39, 2004.

[36] A. Peshkin, "Virtuous subjectivity: In the participant observer's I's," in *Exploring Clinical Methods for Sound Research*, D. Berg and K. K. Smith, Eds. Beverly Hills, CA: Sage, 1985, pp. 124– 135.

[37] E. W. Eisner, *The Enlightened Eye: Qualitative Inquiry and the Enhancement of Educational Practice*. New York: Macmillan, 1991.

[38] J. L. Bishop and M. A. Verleger, "The flipped classroom: A survey of the research," in *Proceedings of the 2013 ASEE National Conference*, Atlanta, GA, 2013.

[39] M. J. Lage, G. J. Platt, and M. Treglia, "Inverting the classroom: A gateway to creating an inclusive learning environment," *The Journal of Economic Education*, vol. 31, no. 1, pp. 30–43, 2000.

[40] R. Dawson, "Twenty dirty tricks to train software engineers," in *Proceedings of the 22nd International Conference on Software Engineering*, ser. ICSE '00. New York, NY, USA: ACM, 2000, pp. 209–218.

[41] A. Campbell and L. Norton, Eds., *Learning, Teaching and Assessing in Higher Education: Developing Reflective Practice*, ser. Teaching in Higher Education. Learning Matters, 2007.

[42] W. Hung, "Theory to reality: a few issues in implementing problem-based learning," *Educational Technology Research and Development*, vol. 59, no. 4, pp. 529–552, 2011.

[43] J.-R. Ruiz-Gallardo, J. L. González-Geraldo, and S. Castaño, "What are our students doing? Workload, time allocation and time management in PBL instruction. A case study in science education," *Teaching and Teacher Education*, vol. 53, pp. 51 – 62, 2016.

[44] V. Le, M. Afshari, and Z. Su, "Compiler validation via equivalence modulo inputs," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI2014)*, 2014, pp. 216–226.

[45] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*, 2nd ed. Prentice Hall, 2014.

[46] P.-L. Poon, H. Liu, and T. Y. Chen, "On the effectiveness of error trapping and metamorphic testing for spreadsheet fault detection." *Journal of Organizational and End User Computing*, in press.

[47] Agile Alliance, "What is agile software development?" [Online]. Available: http://www.agilealliance.org/the-alliance/ what-is-agile/

[48] E. H. Chiriac, "Group work as an incentive for learning – students experiences of group work," *Frontiers in Psychology*, vol. 5, p. 558, 2014.

[49] D. Towey and T. Y. Chen, "Teaching software testing skills: Metamorphic testing as vehicle for creativity and effectiveness in software testing," in *Proceedings of the 2015 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE),* Dec 2015, pp. 161–162.