An Adaptive Delivery Strategy for Teaching Software Testing and Maintenance

Mark Allison
Department of Computer Science,
Engineering and Physics
The University of Michigan at Flint
Flint, MI USA
Email: markalli@umflint.edu

Sui F. Joo
Department of Mathematics
Broward College
Fort Lauderdale, FL USA
Email: sjoo@broward.edu

***Abstract*—Today's classroom and learner cohorts are supported by new and innovative technology in an unprecedented manner. The pace of technological advancements however does present challenges to educators in keeping abreast of the state of the art while concurrently facilitating a learner-certered approach. Topics within software engineering are especially susceptible to this phenomena and demands a more individualized adaptive model to support significant learning. Although there is an abundance of sound theoretical models which may address the challenge, the literature is sparse as to contextualization, application or concrete operation. Within a blended classroom, we have implemented an adaptive approach to address the pace of technology advancements with consideration for the skillsets and declarative knowledge of the learner.**

In this paper we present our approach within a software testing and maintenance course and discuss the lessons learnt. We have based our approach on the scientific grounding of Vygotsky's zone of proximal development and discuss the necessary scaffolding, inherent challenges and present an evaluation based on pre/post testing.

I. Introduction

As the demand increases for educators to incorporate new instructional strategies towards significant learning, we are challenged to revise older pedagogical models to fit the current student cohort and educational platforms. This challenge is exacerbated when considered within the context of software systems. Methodologies and tools to design, implement, test and maintain software are experiencing radical changes to keep pace with rising software complexity and ubiquity. Educators are discovering that lesson plans and activities require constant revision each time the topic is delivered in order for such a course to not fall to obsolescence.

A secondary impetus for adaptation within the design of a strong delivery model is to effectively map coursework to the declarative knowledge and abilities of the learner cohort to optimize engagement and learning. To this end we required an adaptive instructional model which targets core theoretical concepts as learning outcomes; addressing the requirements of the group while sensitive to the peculiarities and experiences of the individual.

An adaptable instructional model allows for variation in the paths to learning outcomes utilizing applicable augmentation to support the individual needs of learners[1]. In a blended classroom with on-ground and on-line elements, virtual learning environments (VLE) such as Blackboard[2] and Moodle [3] may provide such augmentation. VLEs may be supplemented by designated resources and tutorials to allow the learner to acquire knowledge at their own pace. As VLEs facilitate collaboration among learners and dissemination of information, it becomes an suitable platform for relegating issues pertaining to the rapidly changing software field and current approaches via active learning activities.

We designed a course scientifically grounded in strong constructivists principles to engage the student in an active learning environment. To accomplish this we drew significantly on Vygotky's seminal work on the zone of proximal development (ZPD)[4]. The constructionist concept seeks to effectively manage learner development by orchestrating tasks and temporary supportive constructs (scaffolding) to challenge yet not frustrate the learner. The process is highly mindful of the learners' cognitive-affective state. Much of the informational scaffolding to address ZPD in teaching software testing and maintenance was accomplish using a VLE as our enabling technology.

The topic of software testing and maintenance has been neglected in computer science curriculum and there is a significant and growing gap in the knowledge of graduating software engineers[5]. The preponderance of software engineering education focuses on developmental activities while results for industry supports a growing need for addressing maintenance activities. It is with this in mind that we felt it necessary to develop a course which would optimize the learning within the timespan allotted.

In this article we discuss the challenges faced, methodology explored to address said challenges, and lessons learnt in the delivery of the core concepts and current trends. To evaluate the approach we utilized pre and post tests and analyses score difference towards empirical evidence of utility. More specifically our contributions are:

1) An adaptive approach to the design and delivery of a software testing and maintenance course based on a hybrid delivery model (onground and online); and
2) An evaluation based on pre and post test analysis.

The remainder of the paper is organized as follows: Section II discusses the pedagogical underpinnings to the approach and

978-1-4799-6600-4/15/$31.00 ©2015 IEEE

presents software testing and maintenance as an intertwined concept within the software life cycle. Section III overviews our motivation and section IV delves in to the adaptive strategy. We evaluate the approach in Section V and discuss what we have learnt in section VI. We summarize and suggest future paths to teaching and learning with this adaptable model in section VII.

II. BACKGROUND AND RELATED WORK

A. Zone of Proximal Development

The constuctivist theory of knowledge sees learning as a knowledge synthesis activity whereby the individual builds upon a mental schema based on *her/his* prior experiences and interpretation of new concepts [6]. The uniqueness of the individual is a core concept and supports our learner-centric adaptive approach. Social constructivism looks at how social interactions affect learning. One key construct is the theory of the zone of proximal development. Observing children's aptitude towards task completion in collaborative environments, psychologist Lev Vygotsky characterized the phenomena as: *"the distance between the actual developmental level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or in collaboration with more capable peers "* [4].

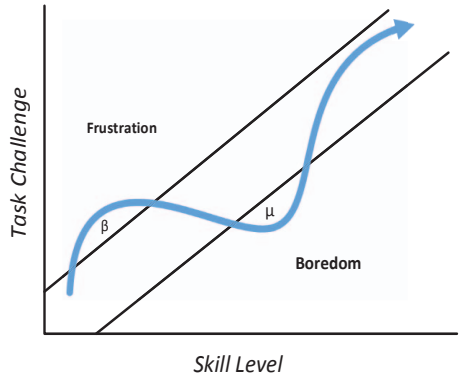

Fig. 1: Zone of Proximal Development

Figure 1 depicts ZPD along spatial dimensions of the difficulty of a task and the skill level of the learner. Within the lower portion, which reflects *boredom*, lies activities which poses a minimal challenge in the context of the learner's skill level. The upper portion, which reflects frustration, indicates that set of tasks for which the learner may have limited success given their skill level. Between these sections lie the ZPD. We may notice that our learner in Figure 1 is outside the ZPD at two points β and μ. At these points the challenge may need to be adjusted and/or supplementation required. The challenge lies in the application of the appropriate scaffolding to maintain the learner within the ZPD.

B. Testing and Maintenance

In order to maintain quality in the face of rising software complexity, singularly focusing on design and developmental best practices is insufficient. The unsung activities of testing and maintenance are key to quality assurance as we move forward in unchartered territory in software engineering. These two intertwined activities are typically responsible for the the majority of resources within the software lifecycle yet have been relegated to a necessary evil within software engineering curriculum .

Testing activities may be responsible for as much of 50% of developmental outlay[5]. Testing is recommended to be integral to all facets of the development, operation and maintenance activities within software[7]. Failure to acknowledge and incorporate proper testing practices has been well documented in the literature and attributed to the demise of a significant percentage of software projects. It is a difficult task to develop an appreciation for testing as a core activity given most students have little or no exposure to the practice in early classroom activities.

In the development of this course we viewed maintenance as those activities undertaken subsequent to the software product delivery. As with testing, maintenance activities have been grossly undervalued and neglected [8]. Maintenance costs associated with software may account for as much as 70% of total life-cycle expenditure [9], yet is primarily conducted by junior engineers as their more senior counterparts aspire to specification and design tasks. We adopt the view of maintenance as an evolutionary development activity. Regression testing is highlighted as a maintenance activity solidifying the bond between the two concepts.

Instilling the relevance of these tasks as direct impacting factors for improving the quality and reducing the cost of software is a reoccurring prescript during the course.

III. MOTIVATION

The approach utilizes adaptable coursework model with fixed learning outcomes. This is a departure from the rigid structured lesson plans which details and timelines all aspects of the course regardless of prior exposure to learning concepts; a *procrustean fit* [1]. The legacy approach of a fixed instructional model assumes:

1) The educator has sufficient knowledge of the skillsets and experiences of the learner a priori to optimize the learning experience.
2) The learning objectives are achievable using a rigid predetermined set of resources that can be identified prior to the beginning of the course and so sufficient as not to require modification or enhancements over the duration of the course.

On both accounts it is difficult to predict these variables especially in the context of software technology wherein tools and methodologies have an extremely limited lifespan. The state of the art may shift dramatically during the span of a course. Our solution is to allow students to explore on their own, once given a deep appreciation for, and grounding in core concepts. For instance, there is a distinct possibility that the testing tools a student may use on their first job after graduation has yet to be conceived or developed at this time.

[1] In Greek mythology, Procrustes owned a bed on which weary passersby were forced to lay. If such a person was too tall their legs were cut shorter and if they were too short then they were summarily stretched; either way the visitor died.

We propose that the skills gained in the exploratory process may have more longevity. The core concepts do however require some experiential learning activities to solidify the constructs. To that end we primarily employed the rich tooling environment of eclipse [10] with JUnit, Findbugs [11], Test and Performance Tools Platform (TPTP)[12], and Jubula test automation [13] as extensions.

Given the affordances of the current digital learning landscape, we are now more capable than ever to retool learning and customize our approach to learner-centric teaching and learning.

IV. Adaptable Delivery Strategy

At the onset, our framework for an adaptable strategy had self-imposed constraints; the adaptation by no means should compromise the intended learning outcomes. Adaptive instruction refers to a flexible strategy for enabling learners with different skill sets and learning abilities to achieve similar outcomes accommodating different learning routes and paces[14][15].

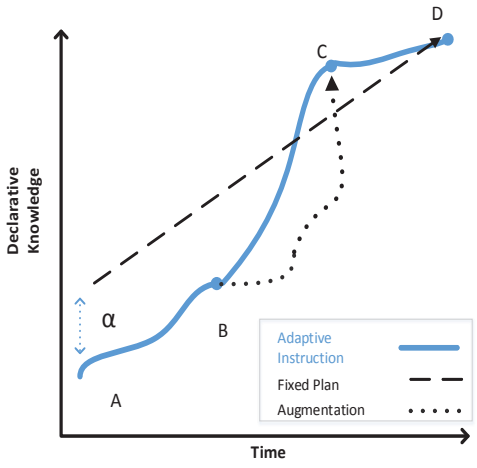

Fig. 2: Adaptive Delivery of Coursework

Figure 2 portrays the adaptive strategy for coursework for the classroom as a unit, measured along the dimensions of increasing declarative knowledge of the subject matter presented, and the duration of the course. The solid line would then represent the rate of knowledge delivery.The broken line represents the inflexible legacy delivery model where all is planned and sequenced in advance. The course begins at A and necessitates an understanding of the skill level of the preponderance of the class. α denotes the difference with the rigid model. It is here that the learner will get turned off from the subject as she has limited context within her mental schema to connect the new concepts. We see that adaptation as time progresses may have different rates of delivery at adaptation points B and C. At B the tasks are too mundane so the coursework may become more challenging. Augmentation (dotted line) is required as some students may need help to keep up with the coursework. At C the coursework becomes too challenging and the educator may need adjust down. The same learning objectives as with the inflexible model are accomplished at D.

A. Groundwork

Critical to the constructivists approach is the learners perception of the relevance of the subject matter and their ability to situate the concepts testing and maintenance effectively within their mental schema of software engineering. We emphasized agile process models to be in line with industry practices. We utilized Brooks's work on the *essence and accidents* relating to software development [16], for learners to gain a deep appreciation for the uniqueness of software as an engineered product. Exposure to the concept of software systems as inherently possessing attributes of invisibility, changeability, conformity and complexity. This became a excellent segway to emphasize change in the face of complexity as a driver of maintenance activities. We present change as originating from user requirements, the environment within which the software operates, and from within the software itself. These are mapped to adaptive, preventative, perfective and corrective maintenance activities. We continued by looking at the bond between testing and maintenance primarily from the perspective of quality assurance. Once the groundwork is laid we look at the supportive aspects in the form of scaffolding.

B. Scaffolding

Scaffolding is the overarching term to represent temporary support constructs in the form of resources and guidance given by the expert (educator) to facilitate the learning process [17]. In this section we discuss such scaffolding which we find most effective within the course.

The student needs to be made aware that she is the at the center, and the driver of the learning process. Fostering intrinsic motivation should preclude and permeate all coursework since our responsibility within higher education is to promote life long learning[18]. It is our position that once we place the learner in charge of the process, the totality of available resources allows for significant learning given sufficient intrinsic motivation is in place. Como et.al.[1] identifies the dichotomy of objectives for adaptation as: (1) aptitude development such as problem solving and computational skills and, (2) declarative knowledge required to proceed with instruction.

To address the first objective, much of testing and maintenance activities require computational thinking and problem solving skill development. Learners were exposed to a systematic way of handling problems. Utilizing a Polya based approach [19], problems are addressed in discrete stages of: understanding the problem; devising a plan; carrying out the plan; then looking back on that pathway to the solution(reflection and revision).

Objective 2 pertains to background knowledge relating to a subject matter. There is a significant amount of declarative knowledge regarding the software engineering process that is required prior to embarking on a study of testing and maintenance principles. If the learner does not possess such knowledge then supplementation is critical per ZPD. Carrier and Jonassen [20] classifies the supplementation for adaptation into four distinct categories: remedial, preferential, compensatory,

and challenging. We focus on the first and last items to keep learners within the ZPD practice. Remedial supplementation is required to ensure students have the background to be able to understand and absorb new concepts. Equally important is the challenging supplementation used for students who may become bored with the pace of the course.

With scaffolding there needs to be a conscious and systematic fading of the support given. Fading is that final step in scaffolding. The educator as facilitator is required to determine the optimal time and pace at which some support in withdrawn and the student internalizes the scaffold. This allows the learner to transition away from the dependency.

C. Challenges and Lessons Learnt

A major challenge was to dispel the stigma of software testing and maintenance as an unholy alliance of arguably the two least favored tasks within the realm of the software life cycle. Images of the bland and mundane are conjectured at the mere mention of either task, so to bring both together in a semester long course would be tantamount to the ideal recipe for lackluster. Students had to be exposed to the significance of these activities by looking at case studies of real world software disasters attributed to poor testing or maintenance activities.

We found that students placed a strong emphasis on extrinsic motivation such as grades. There had to be extensive effort expended to overcome this roadblock to self directed learning. The student needed to understand that by targeting the exam they may lose focus of the bigger picture; acquiring the knowledge and skillsets required of the craft.

Typical of most Information Systems subjects, the intricacies of the subject matter could not be adequately addressed solely via textbooks and required supplemental materials. Students were required to look at current research in testing and maintenance as applied to trending technologies such as mobile and cloud computing.

Significant effort needed to be expended to build supplemental resources into our Blackboard VLE. We found that the students, being millennials, were more responsive to multimedia sources.

Much of the earlier coursework seemed to trivialize the effect of proper developmental practices on maintenance efforts. One activity that we found particularly of interest to the students, involved having them add a feature to another's code. Students found that the prevalent practice of poor documentation, meaningless variable names, and lack of modularity in design makes simple maintenance tasks immeasurably complex; it drives this lesson home.

The concept of Test-Driven Development(TDD) [21], was not well received as it was contrary to the way most of the learners were educated in software development. Since TDD requires some upfront planning, many saw this as an inefficient methodology. It is difficult to see the quality gains on projects on the scale of a one semester course.

V. Evaluation

A. Method

Our sampling consisted of 26 graduate level students. The course presupposed limited knowledge of testing and maintenance a priori, however at that level many of the learners had already been exposed to the concept in one form or another. This created a unique challenge. According to the ZPD, should the pace of the course be too slow then the learners would become bored, conversely if too much knowledge was assumed and the pace was fast then the learners would become increasingly frustrated. Both extremes are to be avoided. To employ dynamism and adaptability there is a requirement for a robust assessment and feedback skeleton. This skeleton comprised formative and summative methods.

Since there was a need to establish the prior knowledge of the of the learners, a pre-test was administered which covered all the learning objectives of the course. A portion of the test required open ended questions which also served to expose greater insights such as the writing skill level and grasp of software vernacular. The results of the pre-test and the correct answers were not discussed as this test also served to measure learning. We administered the same test at the end of the course as the post-test and analyzed the difference.

B. Result

Figure 3 shows the results of our evaluation. The pre-test ranged between 24 and 72, while the post-test ranged between 57 and 100. At left, the pre-test result has a wider scattering from mean compared to post-test results at right. This signifies that the disparity of declarative knowledge pertaining to the course objectives became narrower at the end of the course.

C. Analysis

The test used in our analysis is a matched pairs t-test as two measurements were taken from each individual (pre and post). The sample mean pre-test score was 42.3 and the sample mean post-test score was 83.1. The matched-pairs t-test was performed to test the alternative hypothesis: post-test scores are significantly higher than pre-test scores on average. The test gave a test statistics of t=16.455 with degrees of freedom 29 and P-value 1.5071×10^{-16}. The t-statistic was significant at the .05 critical alpha level. At a 0.01 significance level, we reject the null hypothesis and found that there is enough evidence to conclude post-test scores were significantly higher than pre-test scores on average.

We questioned how much higher would the difference be on average between post-test scores and pre-test scores; therefore we constructed a 95% confidence interval of the mean difference: (35.73, 45.87), this is the interval estimate of the true value of the mean difference between post-test scores and pre-test scores. As we see, there is quite an improvement of test scores.

VI. Discussion

We are pleased with the results of the course. The data supported the engagement observed and formative assessments completed over the course of the class. Since this was the first attempt at adaptive delivery in a blended classroom, we

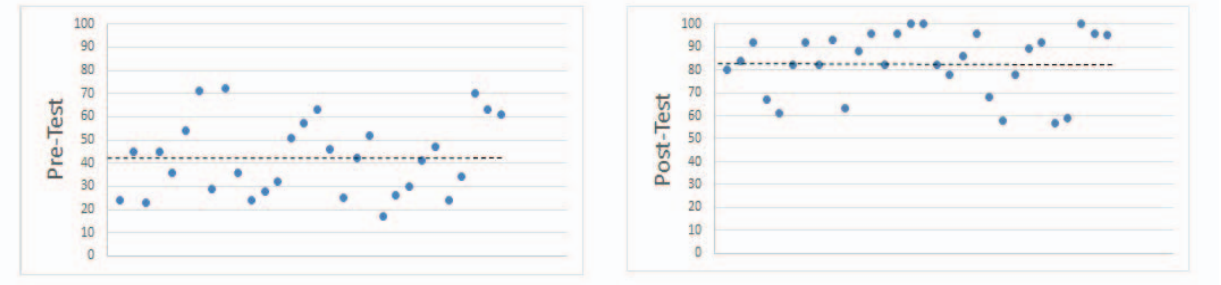

Fig. 3: Pre and Post-Test Distribution

are confident that our method will require less effort for our next adaptive course. We have insufficient evidence that this approach may have similar results for other subject matters, however we will suggest that some level of adaptability and scaffolding are transferable.

A. Threats to Validity

Since the students were asked to volunteer, there were cases where we had pre-test data and no post-test. We filtered this data to only individuals who did both. This may represent a threat to the validity of the data as it may be argued that only student with strong intrinsic motivation would fully participate and would normally do better in the course. In addition, students who dropped the course due to lower initial declarative knowledge would not be represented in this data.

VII. Summary and Future Work

Building and maintaining high quality software is a significant challenge. As educators with constrained time to develop our students to face this challenge, we must look towards methods to leverage the technology and optimize learning. In this paper we presented our adaptable delivery strategy in the context of a course which combines two highly neglected practices within software engineering education; testing and maintenance. We overviewed the challenges, the lessons learnt and presented an analysis of pre-test and post-test scores. The results demonstrated significant average gains between pre and post tests which indicated that learning of the objectives occurred. The experience is as such that we will continue to convert other software related courses towards significant learning gains.

We would like to address other learner aptitude variables such as individual learning styles and motivational states in future coursework.

Acknowledgment

The authors would like to thank The University of Michigan at Flint's Thompson Center for Learning and Teaching for support and constructive feedback pertaining to the intricacies inherent within the course's design and delivery.

References

[1] L. Como and R. Snow, "Adapting teaching to individual differences among immers," *Handbook of research on teaching (3rd ed.). New York: Macmillan*, 1986.

[2] P. Bradford, M. Porciello, N. Balkon, and D. Backus, "The blackboard learning system: The be all and end all in educational instruction?" *Journal of Educational Technology Systems*, vol. 35, no. 3, pp. 301–314, 2007.

[3] M. Dougiamas and P. Taylor, "Moodle: Using learning communities to create an open source course management system," in *World conference on educational multimedia, hypermedia and telecommunications*, vol. 2003, no. 1, 2003, pp. 171–178.

[4] L. S. Vygotsky, "Mind and society: The development of higher mental processes," 1978.

[5] T. Shepard, M. Lamb, and D. Kelly, "More testing should be taught," *Communications of the ACM*, vol. 44, no. 6, pp. 103–108, 2001.

[6] D. H. Jonassen, "Designing constructivist learning environments," *Instructional design theories and models: A new paradigm of instructional theory*, vol. 2, pp. 215–239, 1999.

[7] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *2007 Future of Software Engineering*. IEEE Computer Society, 2007, pp. 85–103.

[8] N. F. Schneidewind, "The state of software maintenance," *Software Engineering, IEEE Transactions on*, no. 3, pp. 303–310, 1987.

[9] B. P. Lientz and E. B. Swanson, "Software maintenance management," 1980.

[10] J. desRivieres and J. Wiegand, "Eclipse: A platform for integrating development tools," *IBM Systems Journal*, vol. 43, no. 2, pp. 371–383, 2004.

[11] N. Ayewah, W. Pugh, J. D. Morgenthaler, J. Penix, and Y. Zhou, "Using findbugs on production software," in *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*. ACM, 2007, pp. 805–806.

[12] T. Eclipse, "Eclipse test & performance tools platform project," 2015.

[13] M. Shtakova, "Evaluation of methods for automated testing in large-scale financial systems," 2012.

[14] M. C. Wang and H. J. Walberg, "Adaptive instruction and classroom time," *American Educational Research Journal*, vol. 20, no. 4, pp. 601–626, 1983.

[15] M. C. Wang, "Adaptive instruction: Building on diversity," *Theory into practice*, vol. 19, no. 2, pp. 122–128, 1980.

[16] F. P. Brooks and N. S. Bullet, "Essence and accidents of software engineering," *IEEE computer*, vol. 20, no. 4, pp. 10–19, 1987.

[17] P. L. Beed, E. M. Hawkins, and C. M. Roller, "Moving learners toward independence: The power of scaffolded instruction," *The Reading Teacher*, pp. 648–655, 1991.

[18] Y.-G. Lin, W. J. McKeachie, and Y. C. Kim, "College student intrinsic and/or extrinsic motivation and learning," *Learning and Individual Differences*, vol. 13, no. 3, pp. 251–258, 2001.

[19] M. A. Allison and S. F. Joo, "Revisiting polyas approach to foster problem solving skill development in software engineers," in *Computer Science & Education (ICCSE), 2014 9th International Conference on*. IEEE, 2014, pp. 379–384.

[20] C. A. Carrier and D. H. Jonassen, "Adapting courseware to accommodate individual differences," *Instructional designs for microcomputer courseware*, pp. 203–226, 1988.

[21] D. Janzen and H. Saiedian, “Test-driven development: Concepts, taxonomy, and future direction,” *Computer*, no. 9, pp. 43–50, 2005.