Carry-on Effect in Extreme Apprenticeship

Hansi Keijonen, Jaakko Kurhila, Arto Vihavainen
Department of Computer Science
University of Helsinki, Finland
{hkeijone, kurhila, avihavai}@cs.helsinki.fi

Abstract—**We argue that the first undergraduate courses are the most important ones on the student's path towards becoming a computer scientist. Therefore, during 2010-2012, we have exercised extensive effort in order to improve the first-semester Computer Science (CS) courses. We have been able to use a learning-by-doing approach called the Extreme Apprenticeship (XA) method accompanied by personal advising even for courses with hundreds of students. We claim that when high demands are met with sufficient support, students learn valuable programming skills that become a foundation that carries them in their further CS courses. In this paper, we analyze how the effects of a three-year effort of renovating our introductory programming courses propagate to further studies. Compared to the control cohorts of 2007-2009, we observe a carry-on-effect caused by the XA method in student success that is visible in the per-student average accumulation of credits after 7 and 13 months after the start of studies. In addition, we can see the effect propagating to mandatory subsequent courses, even without the XA method.**

I. Introduction

It is well known that learning to program is hard [1]. As computer science is typically not taught in high schools, first-year Computer Science (CS) students experience difficulties that are manifested in high failure and drop-out rates in the first programming courses (often referred to as CS1). Improving the operation of the first programming course has been a popular topic for years in the CS education community.

A challenge in examining the improvements in education is that the examination is oftentimes either too broad or too narrow. The administrative view to teaching development emphasizes the student throughput and number of degrees, cost-effectiveness [2], faculty readiness to adopt new teaching methods (see e.g. [3], [4]) but does not elicit the true effect of the improvements in the learning within the CS1 course. On the other hand, a teacher's view and reports thereof emphasize the uniqueness of the course without inspecting the effect on the subsequent courses [5], [6]. Extending the view from a single course is important as the expertise accumulates throughout the degree.

At our department, we have exercised extensive effort on improving the first-semester Computer Science courses that our university students encounter. Due to the teaching improvements and efforts, our department has been awarded various national teaching prizes during the last decade. During the last three years, we have made effort to investigate and apply a learning-by-doing approach accompanied by personal advising even for courses with hundreds of students. The application of the method to our introductory programming courses has both increased students success rates as well as actual learning. The results are significant, as the improvements have been made in a context, where the teaching has already been praised both by the students and the nation.

We purposefully and openly press our students to immerse themselves into a mode of building a strong programming routine by deliberate practise from day one of their studies. Contrary to many other approaches, we do not seek a silver bullet that would allow our students to spend fewer hours on learning. Instead, we want our students to really put the effort into purposefully guided learning-oriented activities. We claim that when high demands are met with sufficient support, students learn valuable hidden skills that become a foundation that carries them during their further studies.

Now, after three years of applying the method to our first semester, we are ready to examine the long-term effect of the *learning* in the students' first CS1 course. It is known that "teacher-induced learning has low persistence, with three-quarters or more fading out within one year" [7]. However, we have observed a rise in the skills of the students that suggests there is a carry-on effect from the first course to subsequent courses.

In this paper, we present the data that shows that the increase in programming skills in the first course enables more students to continue on a path to become a computer scientist. We observe an effect on the success of the studies 7 and 13 months after the initial programming course, as well as the courses immediately after the first programming course.

II. Background and Context

In Finland, there are no tuition fees for anyone in studies in higher education. There are more study positions in STEM (science, technology, engineering, and mathematics) subjects in higher education than there are high school students with a suitable high school course selection, and naturally, every institution wants to recruit good students. Unfortunately Computer Science is not among the most desirable study subjects, therefore, it is not very difficult to secure a study right in CS. The entrance exam for CS studies is based on logical and analytical skills and does not require programming knowledge. The studies start with no expectation of previous knowledge on the subject.

The Department of Computer Science at the University of Helsinki has been selected as a national Centre of Excellence (CoE) in higher education twice in a row, 3 years at a time. This is a remarkable achievement for the department, since in the last round of CoE, the status was awarded only to 10 units in the whole country. The CoE status was received based on ten years of well-documented department-wide teaching improvements, such as formalized study circles, detailed and explicit

978-1-4673-5261-1/13/$31.00 ©2013 IEEE

learning outcome rubrics for all mandatory and steadily recurring courses, and arranging the study environment according to the so-called *constructive alignment* [8]. Thus, it is safe to say that the education and teaching provided by the department has been highly valued and in a solid form already before the advent of the latest development that is in the focus of this paper.

Contrary to the common attitude, where faculty draws away from undergraduate education in order to teach graduate courses and fulfill research demands [9], we have exercised extensive effort to improve the undergraduate education starting from the very first courses. The most recent work in this area has been an improvement to both teaching arrangements and the content of our software engineering-related courses [10].

During the improvement, we have created and started applying a pedagogical method that we call the Extreme Apprenticeship (XA) method. As the first undergraduate courses are important on a student's path towards becoming an expert computer scientist, meaningful support activities must be organized for early courses. Approaching the early student population with too much distance can be detrimental to the learning community, students and teachers alike. Therefore, one important aspect of XA is to reduce the distance between the students and the teachers. In practice, this means constant emphasis on two-way feedback (interaction) between the teachers and the students.

A. *The Extreme Apprenticeship method*

Extreme Apprenticeship (XA) [11] is a method of organizing programming instruction in an effective and scalable manner [12]. It is not only about *learning about expertise* but *becoming an expert* in the practiced skill, e.g. programming. XA is influenced by Extreme Programming [13], where software development best practices such as code reviews are taken to the extreme, and Cognitive Apprenticeship [14], where emphasis is put on making tacit processes visible for the students via modeling, after which the students are scaffolded as they work on the task at hand themselves.

Exercises play a crucial role in XA education. Our courses are carefully structured around collaboratively produced learning objectives and assessment criteria that are visible to the students and teachers alike. Each learning objective is covered using several of exercises, that build on top of each other in a stepwise fashion. The stepwise increment is an adaptation of test-driven development [15] and the Spiral approach in education, where students deepen their knowledge on the topic step-by-step validating their work during each step.

Exercises are designed to help students start easy and deepen their knowledge in an iterative manner. An easy start provides feelings of success and helps students achieve their comfort level. Feelings of success feed the motivation that is known to be fluctuating strongly even within a course [16]. As the students work through the easier exercises, they practice skills that have been relevant in the earlier parts in the course, as well as are introduced to new topics in a gentle fashion. As students proceed within a course, the learning objectives of the exercises start overlapping each other, and more focus is put on facilitating deliberate practise [17].

As students start their work on a task, they first build a mental model of the problem at hand e.g. via well-structured exercise design, process recordings, or during lectures. Once the modeling phase has been continued to a state, where the student feels that she is confident about working on the task, she works on the task under guidance of a more experienced instructor, e.g. a teaching assistant. The teaching assistant scaffolds the student if she needs support, and even in such cases the student is only nudged towards a direction, where she can again proceed on her own.

In our context, the students are constantly helped in computer labs by course instructors, who actively engage the students that work on the exercises; students typically receive help within minutes, depending on the time of day, and the amount of students in the labs. Scaffolding provided by the instructors and learning material are designed to help the students to reach their zone of proximal development [18]. There are tens of weekly exercises, some of which provide step-by-step guidance for completing them, mimicking the solution process that a master can utilize while solving them, while other exercises are open-ended and allow students a larger degree of freedom for designing and programming a solution for them. While the students receive support and guidance in the labs, we also provide (semi-)weekly code reviews for some of the open-ended exercises.

Although there is no limit on the amount of guidance that a student can receive, or on the amount of times that the exercises can be returned, it is of utmost importance that as soon as the student does not require scaffolding and can proceed on their own, the scaffolding is faded, i.e. the support is reduced. The cycle of modeling, scaffolding, and fading takes place several times each week as each week typically contains several learning objectives and tens of exercises; the exercise sets for each week also have a strict deadline, after which they cannot be returned. Even if a solution that the student ends up with is correct, it may still require refinement. Depending on the quality of the solution, the student may be directed to further improve her work in the lab and apply practices such as clean code [19].

In practice, the most significant differences between XA and the traditional operation from the organizational perspective are: 1) there are no lectures in XA (or if there is, the lectures serve the exercises); 2) students are encouraged and expected to use as much time as needed to master the skills (thereby different students use very varying amounts of time in the XA lab during the week). Students are free to come and go as they wish to the XA labs. With careful allocation of resources, XA does not cost more than the traditional way of organizing lecture-based education [12].

So far the results in our XA-based programming education within the programming courses have been impressive. The change from traditional (lecture-based with take-home assignments) has resulted in a statistically significant change in acceptance rates of our programming courses; the average rates of our introductory programming course and advanced programming course have increased by 32% and 37% respectively (see [20] for further details). This is a noteworthy improvement as a lot of effort was already put into the improvement for the introductory programming course.

B. *Students as voluntary TAs*

In XA-based programming courses, most of the work is typically done in computer labs, where the teaching personnel scaffolds the students that work on the exercises. Since starting to apply XA, we have observed a substantial increase in students that are willing to help others in the labs, even on a voluntary basis. As a response to the increase, we have welcomed the student teaching assistants; many of them are in a very early phase of their studies, and are participating in the teaching community even as early as during their second semester [21].

The students that participate as teachers become legitimate peripheral participants [22] of the teaching community. Having young student members as part of the teaching community is beneficial for all parties, as it may increase the retention rate [23] and create a more enjoyable learning context [24]. Between fall 2010 and spring 2013, in addition to the course faculty, we have had 93 junior teaching assistants participating in making the XA experience as positive as possible to the students in the classes. Contrary to some other laudable efforts in using students as agents of educational reform [25], we aim to have some 20% of our students involved in XA labs as TAs.

III. Study Setting

The carry-on-effect of XA-based education is studied using three different measures: (1) grade distribution in the first mandatory programming course; (2) credit accumulation per average student 7 and 13 months after the start of their studies, and (3) success in the expected study path during the first semester by examining the success in two of the subsequent courses right after the first programming course. In all of these examinations, the point of introduction of XA-based education is the year 2010.

The data used in this study is extracted from the official study records of the University of Helsinki, and contains records for students that have enrolled at the university with CS as their major subject since 2007. In total, the database extract has information on 895 students. The yearly intake of students has been aimed at 130 (except 2007 when it was 150). In practice, there is year-by-year fluctuation since a part of the accepted students do not register for CS (as they probably have succeeded in landing a more preferable study place somewhere else). "Overbooking" in the student intake is typically something around 35% but fairly difficult to predict. This is the reason for normalizing the numbers year-by-year so that we can compare the relative, not absolute changes in the results.

The application of Extreme Apprenticeship method for the first courses was started in 2010. Hence, years 2010-2012 are post-XA years, and 2007-2009 are pre-XA years in the data. Pre-XA years means a more traditional way of organizing education around a fixed number of weekly lectures, exercise sessions, and study groups. The teacher responsible for the courses Introduction to Programming and Advanced Programming has been the same during 2007-2011, while a different teacher was assigned to the courses during 2012.

A. *Grade Distribution*

Even though the first programming course has been completely revamped with the advent of XA, the paper-based final exam has been deliberately kept mandatory and as closely corresponding to the exams during the pre-XA era as possible. Therefore, the grade distribution of the course Introduction to Programming is comparable on a yearly basis. It is important to note that the grade distribution at our university is completely decidable by the teacher responsible for the course. In other words, the grades do not need to be forced into a bell curve. Changes in grade distribution can therefore truthfully reflect the changes in student skills and knowledge.

We acknowledge the fact that since XA is about heavy practice, students are likely to accumulate a stronger programming routine and other desirable qualities that are not captured by the traditional final exam. In order to emphasize the thinking and not just the doing – as Allendoerfer et al. [26] aptly put it – a paper-based final exam that requires higher-order thinking skills is a valid addition to the educational arrangements, even when XA is employed.

B. *Credit Accumulation*

Possible differences in credit accumulation were analyzed by extracting the number of computer science credits that each student had gathered in 7 and 13 months since the start of their studies. As there is always a handful of students that do not start their studies during the same year they enroll, we removed the students that had not attempted to take any courses from the analysis.

As the number of students starting their studies each year differs, the credit accumulations have been normalized based on the number of active students, i.e. students, that have at least attempted a single course. After normalization, the results are directly comparable. As the students start their studies on August 1st, the 13 month accumulation for year 2012 is not available during the time of writing this article.

C. *Early Study Path Success*

In addition to the credit accumulation for each student group, we analyze if there is any difference in study path success between students. The student cohorts are built based on the year when they took their first introductory programming course, which is considered as the first step in computer science studies, as it is mandatory for every student.

We analyze two different course pairs for each student cohort: (1) Introduction to Programming and Advanced Programming (both changed to XA after 2010), and (2) Introduction to Programming and Software Modeling (the latter has not been changed to XA). The courses are organized right after the course on Introduction to Programming in the same semester.

The method employed here resembles the research conducted by Carrell and West [27]. However, as the organization of courses and allocation of teachers is not so structured at our department and administratively collected student feedback infrequent, we cannot examine the effects of XA in such a comprehensive fashion.

IV. Data and Results

In this section, we show the data and the results extracted from the study registry. First, we discuss how the grade distribution has changed within our introductory programming course. Then, we focus on the overall credit accumulation between students that have started during different years, and finally, we consider students' early study path success.

It should be noted that there have not been other significant organizational arrangements that can interfere with the results. The required study path for BSc students has been the same from 2007 to 2012. The number of teachers has remained the same, and no differences in student intake can be evoked. However, as the yearly intake in 2007 was 150 and only 130 from 2008 to 2012, we use the year 2008 as a baseline, as one could argue that the student cohort of the year 2007 was somehow inferior to the subsequent years.

Another worthy detail is that all teachers responsible for these courses are tenured teachers, not adjunct or contingent. All of the courses have had several students as paid TAs; the number and the "quality" of TAs is comparable year-by-year.

A. Grade Distribution

The grade distribution in the introductory programming courses from 2007 to 2012 is visible in Figure 1. The areas in dark color, i.e. grade 0, depict the number of students that have failed the course, while the areas in brighter color indicate students that have passed the course. In the XA-based courses, 38.5% of the students have received the highest grade available, i.e. 5, which is indicated by the brightest color. The grade 5 has been awarded to 22.6% of the students in traditional courses. During 2007-2009, 42.3% of the students failed the course Introduction to Programming on the first try, while during 2010-2012, 28.2% of the students failed the course on the first try.

It is clearly visible that the grade distribution and pass rate has been improving. However, as we do not employ tests similar to ACT/SAT, we are not able to directly compare e.g. grade inflation [28], as is possible in several other countries. However, in our context, there are no direct or hidden incentives tied to the grade distribution, and the teacher responsible for every course instance has been a tenured teacher who also teaches many of the subsequent courses to the very same students; "letting the students off easy" would be harmful for the teacher herself.

Successful start on the study path is a valuable first step, as it is a clear signal for a student that she is doing a good job and is appropriately rewarded. A successful first step can start a virtuous cycle for the student but only if the student truly has learned the required skills. Grade inflation would be counterproductive in XA-based education.

B. Credit Accumulation

Credit accumulation describes the number of credits that students have received during an observed interval. The students are grouped based on the year when they enroll at the university and start their studies. The number of credits has been aggregated from the student groups. Table I shows number of students, sum of credits after 7 months of studying

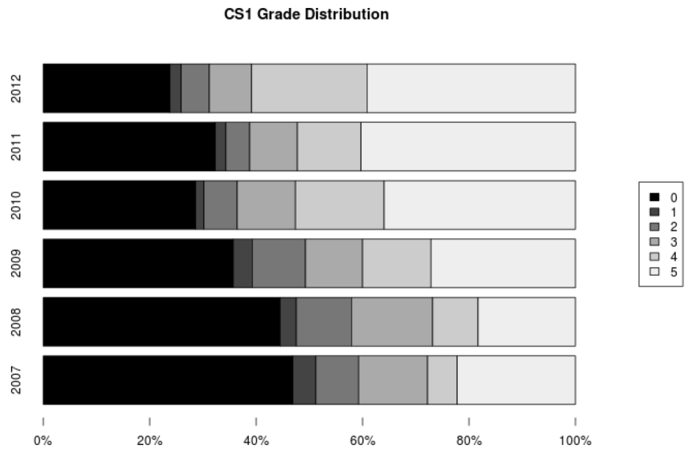

Fig. 1. Grade distribution for the course Introduction to Programming between 2007 and 2012

and credits after 13 months of studying for each student group. In addition to the sum of credits, normalized credit counts and comparison to year 2008 are also shown. The normalization is done based on the student population size, and year 2008 was chosen as a baseline as the student intake was decreased from 2007 by 20 students. Note that the data only contains students that have started their studies, i.e. at least attempted a single course.

TABLE I. Credit accumulation for student groups from different yearly intakes

Year	Students	Credits 7 (norm, scaled %)	Credits 13 (norm, scaled %)
2007	136	1681 (2237, **91.6**)	2558 (3404, **95.2**)
2008	119	1605 (2441, **100**)	2352 (3577, **100**)
2009	120	1616 (2437, **99.9**)	2686 (4051, **113.3**)
2010	136	2030 (2701, **110.7**)	3418 (4549, **127.2**)
2011	140	2287 (2957, **121.1**)	3352 (4334, **121.1**)
2012	168	3042 (3277, **134.3**)	n/a

When looking at the years 2010-2012, we observe a clear increase in the number of credits that students gain during their early studies when compared to the years 2007-2009. The higher number of freshmen in 2012 is explained by an experiment, where we utilized a massive open online course (MOOC) in programming as an entrance exam to CS studies [29]. The number of students that received a study place through "the normal way" was not influenced by the experiment.

To compare whether there is a difference between the 2007-2009 and 2010-2012 cohorts, analysis of variance (ANOVA) was conducted on the credit gains after 7 and 13 months. With three samples in both groups, there is a statistically significant difference between the groups ($p < 0.05$). In the 13 month groups, where the numbers from 2012 is missing, there is no statistically significant difference ($p = 0.062$).

This may be partially explained by the number of samples, and partially by the introduction of XA. XA was introduced during spring 2010, and some of the students that failed first programming courses during fall 2009 retook their programming courses during spring 2010. Typically, spring versions of the programming courses are populated by CS minor students, whereas fall versions are for CS major students.

In addition, we analyze the credit gains of students that attempted their studies, i.e. enrolled to at least a single class, and students that passed courses.

When analyzing the students that attempted their studies, the students in the pre-XA group gained 13.1 credits during the first 7 months (n=375, σ=11.6), while the students in the post-XA group gained 16.6 credits (n=444, σ=11.1). The two groups were also compared using an ANOVA test, which indicated that there is a statistically significant difference for the pre-XA and post-XA groups after 7 months of studies ($p < 0.01$).

When considering the credit gains after 13 months, where year 2012 has been excluded due to currently unavailable data, the pre-XA group gained 20.3 credits on average (n=375, σ=18.6), while the post-XA group gained 24.5 credits on average (n=276, σ=19.0). A statistically significant difference was observed using an ANOVA test ($p < 0.01$).

When considering students that passed courses, i.e. they have passed at least a single course, the students in the pre-XA group gained 17.0 credits during the first 7 months (n=289, σ=10.4), while the students in the post-XA group gained 19.6 credits (n=376, σ=9.3). An ANOVA test indicated that there is a statistically significant difference for the pre-XA and post-XA groups after 7 months of studies ($p < 0.01$).

When considering the credit gains after 13 months, where year 2012 has been excluded due to currently unavailable data, the pre-XA group gained 25.6 credits on average (n=297, σ=17.4), while the post-XA group gained 29.3 credits on average (n=231, σ=17.1). A statistically significant difference was observed using an ANOVA test ($p < 0.05$).

C. Early Study Path Success

In order to validate the effect of XA, we want to examine the mandatory first course (Introduction to Programming) and see whether the success in two mandatory subsequent courses (Advanced Programming and Software Modeling) benefits from the fact that the first course is based on XA or not. To validate the effect even further, one of the subsequent courses (Advanced Programming) is also XA-based, but the other one (Software Modeling) is not. All of these three courses are mandatory courses for every BSc student in CS, and all of the three courses are scheduled to be taken during the first semester.

In the following examination, study path success describes the student percentage that has succeeded in a specific course pair on the first attempt. The percentage for yearly course pairs is shown in Table II.

TABLE II. STUDY PATH SUCCESS WHEN MOVING FROM INTRODUCTION TO PROGRAMMING TO ADVANCED PROGRAMMING AND INTRODUCTION TO PROGRAMMING TO SOFTWARE MODELING

Year	Intr. Prg. & Adv. Prg	Intr. Prg. & SW. Modeling
2007	45.1	41.5
2008	39.2	48.8
2009	50	54.2
2010	68.5	63
2011	71.1	74.4
2012	70.3	72.2

Before XA, the year with the best success was 2009, where 50% of the students that enrolled in both Introduction to Programming and Advanced Programming passed both courses on their first attempt. A similar result is visible in the course pair Introduction to Programming and Software Modeling; 54.2% of the students that started both courses passed both on their first attempt.

The lowest scores after the introduction of XA are from the year 2010. Here 68.5% of the students passed both programming courses on their first attempt, and 63% of the students passed both Introduction to Programming and Software Modeling. However, a clear difference can be observed between the pre-XA and post-XA courses. An interesting issue is that the effect of the first course using XA appears to carry over to the subsequent course, irrespective of the use of XA in the subsequent course.

When conducting an ANOVA test for the course pair Introduction to Programming and Advanced Programming, there is a statistically significant difference ($p < 0.01$) between the pre-XA and post-XA groups. Statistically significant difference is observed also for the course pair Introduction to Programming and Software Modeling ($p < 0.05$).

V. DISCUSSION

We have described results from a long-term study of student performance before and after applying a method called XA in our early programming courses. Our results indicate that the grade distribution, pass-rate, overall credit accumulation, and student success in staying on the desired study path have all improved after applying XA, when looking at students' performance after 7 months and 13 months of studying.

At our university, the teachers are not rewarded for performing well, nor are they punished for performing poorly. As a matter of fact, it is very difficult for a teacher to even know how they are performing, as there are no formal measures other than student grades that the teacher herself decides. The administration oversees the study progress on a larger scale but has no measures to evaluate *learning*. Many times even the formal course feedback received from the students has little impact as the teacher herself is the main actor in processing the feedback. Fortunately, in our context where teaching is valued and teachers want to be good teachers rather than bad teachers, our introductory programming courses have always received excellent feedback and been held in high esteem. Therefore, the improvements described in this paper are valuable, as they extend beyond a single course and improve an already well-functional educational arrangement.

The results and XA that are described in this paper are a part of a larger change, which was started in late 2009. Every change requires people willing to change; we have been lucky to have teachers eager to deliberately practice and hone both their skills and knowledge, and apply their knowledge fully into teaching. Communication and bi-directional feedback valued by XA requires that the teachers are on the same level as the students, as students work on their exercises: both receive feedback on what they are doing well, and what could be improved, which allows a cycle, where the courses can be aligned to match the needs of individual learners.

Acknowledgements

We gratefully acknowledge Matti Luukkainen and Arto Wikla, who have both done great work in improving the programming education at our department. We also gratefully acknowledge the grant from the Dean of Faculty of Science for this project. Part of the research is funded by the Centennial Foundation of the Finnish Technology Industries.

References

[1] A. Robins, J. Rountree, and N. Rountree, "Learning and teaching programming: A review and discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, 2003.

[2] E. P. Bettinger and B. T. Long, "Does cheaper mean better? the impact of using adjunct instructors on student outcomes," *The Review of Economics and Statistics*, vol. 92, no. 3, pp. 598–613, 2010.

[3] D. L. Soldan, W. P. Osborne, and D. Gruenbacher, "Modeling the economic cost of inadequate teaching and mentoring," in *Frontiers in Education Conference (FIE), 2010 IEEE*. IEEE, 2010, pp. F3J–1.

[4] S. Fincher, B. Richards, J. Finlay, H. Sharp, and I. Falconer, "Stories of change: How educators change their practice," in *Frontiers in Education Conference (FIE), 2012*, 2012, pp. 1–6.

[5] P. Lasserre and C. Szostak, "Effects of team-based learning on a CS1 course," in *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education*, 2011, pp. 133–137.

[6] J. D. Bayliss and S. Strout, "Games as a "flavor" of CS1," in *Proceedings of the 37th SIGCSE technical symposium on Computer science education*, ser. SIGCSE '06. ACM, 2006, pp. 500–504. [Online]. Available: http://doi.acm.org/10.1145/1121341.1121498

[7] B. A. Jacob, L. Lefgren, and D. P. Sims, "The persistence of teacher-induced learning," *Journal of Human Resources*, vol. 45, no. 4, pp. 915–943, 2010.

[8] J. Biggs, "Enhancing teaching through constructive alignment," *Higher education*, vol. 32, no. 3, pp. 347–364, 1996.

[9] D. M. Shannon, D. J. Twale, and M. S. Moore, "TA teaching effectiveness: The impact of training and teaching experience," *Journal of Higher Education*, pp. 440–466, 1998.

[10] M. Luukkainen, A. Vihavainen, and T. Vikberg, "Three years of design-based research to reform a software engineering curriculum," in *Proceedings of the 13th annual conference on Information technology education*. ACM, 2012, pp. 209–214.

[11] A. Vihavainen, M. Paksula, and M. Luukkainen, "Extreme apprenticeship method in teaching programming for beginners," in *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 2011, pp. 93–98.

[12] J. Kurhila and A. Vihavainen, "Management, structures and tools to scale up personal advising in large programming courses," in *Proceedings of the 2011 conference on Information technology education*. ACM, 2011, pp. 3–8.

[13] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2nd Edition)*. Addison-Wesley Professional, 2004.

[14] A. Collins *et al.*, "Cognitive apprenticeship: Making things visible." *American Educator: The Professional Journal of the American Federation of Teachers*, vol. 15, no. 3, pp. 6–11, 1991.

[15] K. Beck, *Test driven development: By example*. Addison-Wesley Professional, 2003.

[16] A. Dillon and J. Stolk, "The students are unstable! cluster analysis of motivation and early implications for educational research and practice," in *Frontiers in Education Conference (FIE), 2012*, 2012, pp. 1–6.

[17] K. A. Ericsson, R. T. Krampe, and C. Tesch-Römer, "The role of deliberate practice in the acquisition of expert performance." *Psychological review*, vol. 100, no. 3, p. 363, 1993.

[18] L. Vygotsky, "Mind in society," 1978.

[19] R. C. Martin, *Clean code: a handbook of agile software craftsmanship*. Prentice Hall, 2008.

[20] A. Vihavainen and M. Luukkainen, "Results from a three-year transition to the extreme apprenticeship method," *Proceedings of the 13th IEEE International Conference on Advanced Learning Technologies*, July 2013.

[21] A. Vihavainen, T. Vikberg, M. Luukkainen, and J. Kurhila, "Massive increase in eager TAs: experiences from extreme apprenticeship-based CS1," in *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, ser. ITiCSE '13. New York, NY, USA: ACM, 2013, pp. 123–128. [Online]. Available: http://doi.acm.org/10.1145/2462476.2462508

[22] J. Lave and E. Wenger, *Situated learning: Legitimate peripheral participation*. Cambridge university press, 1991.

[23] C. Stewart-Gardiner, "Improving the student success and retention of under achiever students in introductory computer science," *Journal of Computing Sciences in Colleges*, vol. 26, no. 6, pp. 16–22, 2011.

[24] P. E. Dickson, "Using undergraduate teaching assistants in a small college environment," in *Proceedings of the 42nd ACM technical symposium on Computer science education*. ACM, 2011, pp. 75–80.

[25] G. Herman, K. Trenshaw, and L.-M. Rosu, "Work in progress: Empowering teaching assistants to become agents of education reform," in *Frontiers in Education Conference (FIE), 2012*, 2012, pp. 1–2.

[26] C. Allendoerfer, M. Kim, E. Burpee, D. Wilson, and R. Bates, "Awareness of and receptiveness to active learning strategies among stem faculty," in *Frontiers in Education Conference (FIE), 2012*, 2012, pp. 1–6.

[27] S. E. Carrell and J. E. West, "Does professor quality matter? evidence from random assignment of students to professors," *Journal of Political Economy*, vol. 118, no. 3, 2010.

[28] I. Y. Johnson, "Contingent instructors and student outcomes: An artifact or a fact?" *Research in Higher Education*, vol. 52, no. 8, pp. 761–785, 2011.

[29] A. Vihavainen, M. Luukkainen, and J. Kurhila, "Multi-faceted support for MOOC in programming," in *Proceedings of the 13th annual conference on Information technology education*. ACM, 2012, pp. 171–176.