Extreme Apprenticeship Method: Key Practices and Upward Scalability

Arto Vihavainen, Matti Paksula, Matti Luukkainen and Jaakko Kurhila
University of Helsinki
Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
Fi-00014 University of Helsinki
{ avihavai, mpaksula, mluukkai, kurhila }@cs.helsinki.fi

Final draft

Originally appeared as: A.Vihavainen, M.Paksula, M.Luukkainen and J. Kurhila: Extreme apprenticeship method: key practices and upward scalability. In ITiCSE 2011: Proceedings of the 16th annual conference on Innovation and technology in computer science education. ACM Press, 2011.

Abstract

Programming is a craft that can be efficiently learned from people who already master it. Our previous work introduced a teaching method we call *Extreme Apprenticeship* (XA), an extension to the cognitive apprenticeship model. XA is based on a set of values that emphasize doing and best programming practices, together with *continuous feedback* between the master and the apprentice. Most importantly, XA is individual instruction that can be applied even in large courses. Our initial experiments (n = 67 and 44) resulted in a significant increase in student achievement level compared to previous courses. In this paper, we reinforce the validity of XA by larger samples (n = 192 and 147) and a different lecturer. The results were similarly successful and show that the application of XA can easily suffer if the core values are not fully adhered to.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education *Computer Science Education*

General Terms
Human Factors

Keywords
cognitive apprenticeship, continuous feedback, instructional design, programming education, best practices

1 Introduction

Arts *and* crafts are necessary components in programming excellence. Unfortunately, it is a "public secret" among many respected programming professionals that university programming courses can be counterproductive as non-optimal tools and old academic programming practices must be unlearned in order to develop competence in coding quality. Part of the problem is that learning programming is hard (e.g. [3, 12, 11, 10]).

Although a long line of research [13, 16, 11] indicates that the problem is not in learning the syntax or semantics of individual language constructs, but in mastering the process on how to combine constructs into appropriate programs, lectures still tend to be structured according to the language constructs, rather than the more general application strategies.

Nowadays, many admit that lecturing is not the best way to support learning to program but even exercise sessions tend to remain as methods to just witness achieved learning results; typically, there is only minimal guidance to the students doing the exercises. Accumulation of applied knowledge in a student is assumed and expected. However, it is well-established in educational psychology (e.g. [7]) that, due to human cognition, a minimally-guided approach can be considered sub-optimal for novices learning tasks such as programming.

In a recent paper we presented a method that radically alters the traditional way to approach introductory programming (CS1) education in the context of formal higher education called *Extreme Apprenticeship* (XA) [14]. XA is based on a set of values and practices that emphasize *actual doing* of relevant work together with *continuous feedback* as the most efficient means for appropriate learning. Our previous experiments consisted of 67 and 44 students. The results were strongly encouraging, so we continued the experiment with significantly larger (n = 192 and 147) samples.

As the XA method strives to be as lecturer-independent as possible, this time the XA experiment was conducted with a separately recruited lecturer with extensive experience in traditional lecture-based courses but no involvement in the development of the XA method. In addition, as the XA method is by definition direct one-on-one interaction between the master and the apprentice, a question of scalability of the method becomes important. We briefly describe the tools and organizational approach used to scale up the XA method to larger courses.

2 Extreme Apprenticeship

The Cognitive Apprenticeship (CA) model [5, 6] has many applications in teaching programming (see e. g. [1, 2, 4, 8]). Extreme Apprenticeship (XA) builds on the Cognitive Apprenticeship model. Similarly, it emphasizes the process and consists of three phases: modeling, scaffolding and fading.

In modeling, the master, a teacher or an instructor, arms the apprentice, a student, with a conceptual model of the process. An effective conceptual

model is a set of *worked examples* [4], i.e. a detailed description of completing a programming task from start to finish. While completing the task, the master is thinking aloud all the time, explaining the decisions made during the process.

After acquiring the conceptual model, apprentices are exposed to tasks (i.e. exercises) to be completed under the guidance of a master. Scaffolding refers to supporting apprentices in a way that they are not given answers, rather, just enough hints to be able to discover the answers to their questions themselves. Scaffolding works especially well if apprentices are in the zone of proximal development as described by Vygotsky [15].

Fading of scaffolding occurs when the apprentice starts to master a task.

Even though XA builds upon CA, it differs significantly from many recent applications of CA in teaching programming (compared to e.g. [1, 2, 4, 8]). Our XA method is described by its core values that should be stressed in all course activities (paraphrasing differs slightly from the original description in [14]):

- The craft can only be mastered by actually practicing it. The skills to be learned are practiced as long as it takes for each individual.
- Continuous feedback flows in both directions. The apprentice receives feedback about his/her progress, and the master receives feedback by monitoring the successes and challenges of the apprentices.

The values above induce a set of practices to be applied in all courses:

1. Effectiveness of lectures in teaching programming is questionable; therefore, lecturing should cover only the minimum before starting with the exercises.
2. Topics covered in the lectures have to be relevant for the exercises.
3. Exercises start early, right after the first lecture of the course. During the first weeks of the course all the apprentices are already solving an extensive amount of simple exercises. This gives all the apprentices a strong routine in writing code and a motivational boost right at the start of the course.
4. Exercises are completed in a lab in the presence of masters scaffolding the instruction. There must be ample time to complete exercises while masters are present.
5. Exercises are split into small, achievable tasks. These small intermediate steps guarantee that apprentices can actually see that their learning is progressing.
6. Exercises are the driving force, so the majority of exercises are mandatory for all the apprentices.
7. The number of exercises should be high and to some extent repetitive in their nature.
8. Exercises have to provide clear guidelines, i.e. starting points and structures, e.g. on how to start solving the task and when a task is considered finished.

9. While doing the exercises apprentices are also encouraged to find out things that are not covered during the instruction provided.
10. Best up-to-date programming practices are emphasized throughout the scaffolding phase as they can be incorporated into instruction without any extra effort.

3 Courses in Spring and Fall 2010

Our semester-length (14 calender weeks) CS1-type introductory Java programming course consists of two separate parts: *Introduction to Programming* and *Advanced Programming*. Topics covered in Introduction to programming are assignment, expressions, terminal input and output, basic control structures, classes, objects, methods, arrays and strings. Advanced programming concentrates on advanced object-oriented features such as inheritance, interfaces and polymorphism, and discusses the most essential features of Java API, exceptions, file I/O and GUI.

Staffing

In Spring 2010, the lecturer was one of instructors in computer labs scaffolding students in XA-based exercises. Lecture material and exercises were aligned to suit scaffolding.

In Fall 2010, the lecturer was not a part of the XA team, in other words, he was not scaffolding students in the lab using XA method. Scaffolding was conducted solely by a group of instructors. All but one instructor were students themselves. In both courses, all instructors were compensated 17 Euros per hr, typically 2-6 hrs weekly.

There was an implicit hierarchy with the instructors, as a more experienced teacher acted as the instructor coordinator and was responsible for recruitment of additional instructors. A few instructors had significant programming experience (but limited teaching experience). Many of the instructors were in the early stages of their studies, their teaching experiences were limited to student tutoring at most. Some were truly novice programmers as they did not have any programming experience outside the few courses they had just passed at the university. The only common denominator among the instructors was the attitude: ready to confront the students in person and their challenges; active and eager to help.

Instructors were recruited "on the fly". As good atmosphere in the scaffolding sessions became a talking point within informal student communities, many students volunteered to be part of the XA-based course implementation.

Each instructor had the possibility to choose the most preferable time slots for him or her. Instructors were also able to call for more help on demand via IRC or text messaging, as some other instructors in the instructor pool were available to join in on short notice. We allowed double-teaming at the times we knew the labs were going to be full. We aimed for a 1 per 10 instructor per student ratio in the lab.

Study material and lectures

The study material (including lectures) play a key role in the modeling phase in teaching the skills to be learned. On the other hand, as programming is a craft, it requires plenty of practice. In Spring 2010, we reduced the number of lectures from the usual 5 hours per week to just 2 hours. In Fall 2010, the head lecturer reduced his lecture hours from 5 to 4 per week.

All the material shown in the lectures was available to students on-line as a web page, written in book-like format. The material followed the structure of exercises, allowing students to read the material as they proceeded.

Exercises

It is expected that students attending XA-based courses use most of the time they devote to the course in active solving of programming exercises. This trains the routine and gives a constant feeling of success by achieving small goals. Exercises especially in the beginning of the course were aimed to build up programming routine and confidence, as well as getting familiar with the environment and tools.

For each week we introduced a set of new exercises, an amount ranging from 15 to almost 40. Most of the initial exercises were small, especially at the start of the course, like "output numbers from 1 to 99". Sequential small exercises combined into bigger programs. Composing bigger programs showed students how to split a big task to sub-tasks – a vital skill in programming. Many of the sequentially done small exercises ended up as relatively large projects.

Exercise Sessions

All exercise sessions were organized in computer labs where students worked to solve the exercises. Help, in form of instructors, was continuously available during exercise sessions. Anyone could enter the lab without having to reserve a specific time slot. In Spring 2010, each week had 8 hours of exercise sessions; in Fall 2010, each week had 20 hours. Students were free to attend as many sessions as needed.

An important principle in our approach is that actual programming starts as early as possible. The first exercise session time was right after the starting lecture of the course. For the first week the students already had 30 small exercises to solve. Due to the guidance available, even those with no previous experience in programming managed well.

In order to enforce good programming habits such as principles of Clean Code [9], students had to have their finished solutions accepted by the instructors. If an instructor noticed a flaw in the approach (bad naming or indentation, too long methods, classes with to many responsibilities, too complex solution logic for the problem, etc.), he pointed it out, and the student had to redo parts of the exercise. In general, we allowed no compromises in the solutions of students. This way, each student refined their solutions to the point where the solutions could be regarded as "model answers".

Continuous Feedback

During the course we implemented continuous feedback to provide fast evaluation and a continuous feeling of progress for the students. During the exercise sessions students received positive reinforcement from the instructors.

If a student did not have specific questions during the exercise session, the instructors were active in making sure (in a non-intrusive fashion) that they were working towards the right direction with good working habits. If something to correct was noticed, the instructor nudged the student to the right direction by asking a question about the approach or by providing constructive feedback. This was the key continuous feedback as the hints received during the learning process are essential for acquiring good programming and problem-solving habits. Instructors were not allowed to give direct solutions to the exercises, and the key idea was to support the students so that they could figure out the solutions themselves.

In addition to instructor feedback, students had their completed exercises marked down to a check-list, allowing them to see the check-list filling with marked exercises. We feel that the list played an important role in feedback; every check was a small victory. Check-lists were also updated to the course web-page at the end of every day, allowing students to see the progress of other students as well.

The final exams both in Spring and Fall 2010 were constructed to be as similar as possible to the usual programming exams conducted at our university to provide meaningful comparison of the course results. The exams were programming on paper. A student had to get 50 % of the total maximum score in order to pass the course, regardless of the number of exercises finished in the XA scaffolding.

4 Results

4.1 Incremental validation of Extreme Apprenticeship

The introductory programming courses at the Department of Computer Science, University of Helsinki are taught during both fall and spring semesters. Fall semesters consist mostly of CS majors, while Spring semesters consist mostly of CS minors.

Before Spring 2010 both the programming courses have followed very traditional teaching model based on lecturing and take-home exercises with weekly exercise sessions. The first course implementing Extreme Apprenticeship method was held during Spring semester 2010.

Next we will compare the outcome of the Extreme Apprenticeship-based courses to the previous course instances from past 8 years in terms of percentage of passed students. The results are reported separately in the tables below for Introduction to programming and Advanced programming. The XA-based implementations are highlighted in bold face. The column titled n denotes the

number of students in each course. The numbers are comparable for all the course implementations as the exams have been alike.

Introduction to Programming

	n	passed
s02	92	38.0 %
f02	332	53.6 %
s03	98	39.8 %
f03	261	64.0 %
s04	84	61.9 %
f04	211	59.2 %
s05	112	46.4 %
f05	146	54.1 %
s06	105	41.9 %
f06	182	65.4 %
s07	84	53.6 %
f07	162	53.0 %
s08	72	58.3 %
f08	164	56.1 %
s09	53	47.7 %
f09	140	64.3 %
s10	**67**	**70.1 %**
f10	**192**	**71.3 %**

Advanced Programming

	n	passed
s02	88	26.1 %
f02	249	56.2 %
s03	65	30.8 %
f03	228	59.2 %
s04	66	43.9 %
f04	177	66.1 %
s05	70	57.1 %
f05	125	56.0 %
s06	52	44.2 %
f06	147	67.3 %
s07	53	58.5 %
f07	136	59.6 %
s08	29	51.7 %
f08	147	56.5 %
s09	22	50.0 %
f09	121	60.3 %
s10	**44**	**86.4 %**
f10	**147**	**77.6 %**

The long-term average (excluding Spring and Fall 2010) for passed students in Fall semesters is 58.5% and in Spring semesters 43.7%. In Spring terms, most of the participants are computer science minors. As can be seen, the result in Spring 2010 was higher than it has previously been, 70.1% , the second highest pass-rate being 65.4%. In Fall 2010 it was 71.3%, which was even better than the results from the first XA implementation in Spring 2010.

The pass-rate from Spring and Fall 2010 are approximately the same: 70.1% and 71.3%. Previous course instances have had a clear long-term difference in the failure rates between Fall and Spring semesters, which we do not observe in the XA-based courses. One explanation is the start-early approach: early success brings motivation.

The trend in the Advanced programming course is similar: the average passing percentage in Fall terms is 60.1% and Spring terms 45.3%, both marginally higher than the pass-percentages for the introductory course. This can be explained by the fact that students failing the Introductory course do not take part in Advanced programming.

The acceptance percentage in Spring 2010 was 86.4%, an all-time high in the department with a clear margin. The most natural explanation for the remarkably high passing rate is that the programming routine built during normal course implementations has been quite fragile for an average or below-average student. In an XA-based course, those students who survived from the initial

shock of Introduction to Programming were improving all the time.

In Fall 2010, Advanced programming course pass-rate continued on a remarkably high level: 77.6%. Although not as high as expected, it is significantly higher than the average in previous, traditional Advanced programming courses. The reasons for less-than-expected increase in pass-rate are discussed in the next section.

4.2 Teacher-independence in Extreme Apprenticeship

Instructors spend a lot of time together in the labs and in informal interaction networks (mainly IRC and face-to-face discussions). In a way, this interaction means that instructors are mentoring each other. We observed that new instructors – invited to serve as instructors – were ready to scaffold without any training. Invitation was considered an honor.

Dynamic, on-demand allocation of instructors and the interaction between them brought implicit roles among them. This was evident during the courses when the responsibilities and time spent in the labs for less-experienced instructors faded towards the end, while more experienced stepped up. The same was true for all the courses in Spring 2010 and Fall 2010, even though the number of instructors was scaled up nearly threefold.

Although the result was very good (77.6%) when compared to the previous courses, it did not match our high expectations based on the experiences from Spring 2010 course (86.4%). The results were perplexing particularly since many of the instructors were the same for Spring and Fall 2010, thus having more experience in scaffolding. Moreover, in Fall 2010 the lecturer was one of the most lauded teachers at the whole university, having decades of experience in lecturing CS1-type programming courses.

This led to an investigation in the exercises and course material. The ultimate control (and thus the ultimate responsibility) over the course was handled by a lecturer not participating or involving himself in XA-style scaffolding in the computer labs. The lecturer's responsibilities covered generating the exercises as well. Even though the lecturer received the exercises and material from the Spring 2010 course, it seems that the importance of XA-style exercises was not fully conveyed to the lecturer – the exercises need to be relevant and there must be enough exercises for students to do. Introduction to Programming in Fall 2010 followed mostly the exercise sets from Spring 2010, but the exercises in Advanced programming in Fall 2010 converged towards a more traditional course implementation. The lecturer argued that the exercises and their contents are too demanding for the course. Therefore, he chose to use quite a bit of his old material instead, and reduced the number of mandatory exercises. For example, the first week of Advanced programming in Spring 2010 had 34 exercises, while the corresponding week in Fall had 11 exercises. The trend was similar throughout the whole Advanced programming course.

In Spring 2010, the number of required exercises was much higher than in Fall 2010. It is easy to see from the weekly accumulated data that the students

stop challenging themselves if the course structures are not encouraging them continue.

When we reflect the activities during the Advanced programming course in Fall 2010, we can identify which XA practices were overlooked in: (2) Topics covered in the lectures were not always relevant for the exercises; (7) The number of required exercises was not high enough; (8) Exercises did not provide clear guidelines, i.e. starting points and structures. In short, part of the exercises were not XA.

As the effect of lecturing is small (many did skip the lectures), the problems with practices 7 and 8 are far more serious. Nevertheless, the sole reason for all the problems mentioned above was the lack of bi-directional flow of information between *every* participant in the process. Instructors scaffolding in computer labs received implicit and explicit feedback directly from the students and the challenges they faced. The lecturer, not present in scaffolding, did not receive explicit feedback from the labs. In this context, explicit feedback means that the lecturer did not *directly* observe how the students coped with the exercises, and their need for more meaningful and demanding exercises.

4.3 Scalability of Extreme Apprenticeship

XA-style instruction can potentially be overly expensive. We purposefully aimed not to spend more than in our traditional, lecture-driven, guidance-deprived teaching model. On-demand service was ensured by using IRC. On situations where there were too many students, the instructor could ask for extra assistance on-line. The communication tool worked also as a fast way to help and share information on problems that a specific instructor himself had faced earlier – similarly the instructors were able ask for tips on problems they could not help to solve. Self-organization and openness was important, as the mentors themselves handled the laboratories, and it was important to also share the problems with the group.

For bookkeeping of student exercises and allocation of instructors, we utilized online spreadsheets in Google Docs with our own macros. Each instructor was required to mark down the hours at the end of the day, which allowed us to keep track of the money spent so far and the demand for instructors during specific times. We extended our spreadsheet so that it created automatic predictions for the upcoming week based on the previous week. This allowed us to focus resource allocation for rush hours, and on the other hand we were also able to re-assign instructors from not-so-crowded hours. In the end, the price tag for the courses was indeed relatively the same as it has been over the past years.

5 Conclusions

Extreme Apprenticeship provides a solid structure for teaching skills that aim to build good routines and learning best practices from those who already master them. Emphasizing scaffolding in combination with the core values and the

derived practices yields excellent learning results as was seen already in our initial implementations in Spring 2010. The excellent results were repeated in Fall 2010 with three-fold increase in student population.

XA strives to be a method for organizing teaching regardless of personal traits. We examined this aspect of XA by implementing the second cycle of programming courses with XA-style scaffolding under a traditional lecture course. The lecturer was informed about XA and its values and practices but chose not to be involved in scaffolding in the computer labs.

Even though the results for the latest course (Advanced programming) using XA were very good, they were clearly below what was expected. An inspection revealed that important XA-practices were overlooked, due to lack of observed feedback from the labs. Therefore, the approach used in Advanced programming course held in Fall 2010 is an example of an *Extreme Apprenticeship But* method, as in "We used Extreme Apprenticeship, but..". We believe that the ideas behind the Extreme Apprenticeship method – especially continuous bi-directional feedback and scaffolding – should be followed vigorously in order to provide enough support to help novices, struggling to learn programming, to become truly professional programmers.

Anonymous student feedback collected at the end of the courses showed clearly that the students truly valued the XA-style scaffolding. The most convincing evidence came from students that had tried to pass the course earlier in its traditional form, and now received the "XA-experience".

The outcomes from our initial experiments have led us to believe that the role of the lectures in a programming education clearly diminishes if the exercises are properly designed and there is personal support when trying to complete them. Personal support does not need to be overwhelming; instead, even few minutes spent well to nudge a student into right direction by an instructor who is not afraid to confront the student at his or her own level makes a huge difference.

6 Acknowledgments

We acknowledge all the journeymen who have contributed to CS1 education at Helsinki University by helping out in labs and by spreading the good word on joy of programming. Especially, we would like to thank Thomas “Wilhelmsson” Vikberg, who has prototyped XA at the department of Mathematics and Statistics during Spring 2011, and shown that XA is applicable to other domains as well.

References

[1] O. Astrachan and D. Reed. AAA and CS 1: the applied apprenticeship approach to CS 1. In *SIGCSE '95: Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education*, pages 1–5. ACM, 1995.

[2] T. R. Black. Helping novice programming students succeed. *J. Comput. Small Coll.*, 22(2):109–114, 2006.

[3] R. E. Bruhn and P. J. Burton. An approach to teaching java using computers. *SIGCSE Bull.*, 35(4):94–99, 2003.

[4] M. E. Caspersen and J. Bennedsen. Instructional design of a programming course: a learning theoretic approach. In *ICER '07: Proceedings of the third international workshop on Computing education research*, pages 111–122. ACM, 2007.

[5] A. Collins, J. Brown, and S. Newman. Cognitive apprenticeship: Teaching the craft of reading, writing and mathematics. In *Knowing, Learning and Instruction: Essays in honor of Robert Glaser*. Hillside, 1989.

[6] A. Collins, J. S. Brown, and A. Holum. Cognitive apprenticeship: making thinking visible. *American Educator*, 6:38–46, 1991.

[7] P. A. Kirschner, J. Sweller, and R. E. Clark. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, problem-based, experiental, and inquiry-based teaching. *Educational Psychologist*, 41(2):75–86, 2006.

[8] M. Kölling and D. J. Barnes. Enhancing apprentice-based learning of java. In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 286–290. ACM, 2004.

[9] R. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[10] A. Pears, S. Seidman, L. Malmi, L. Mannila, E. Adams, J. Bennedsen, M. Devlin, and J. Paterson. A survey of literature on the teaching of introductory programming. In *ITiCSE-WGR '07: Working group reports on ITiCSE on Innovation and technology in computer science education*, pages 204–223. ACM, 2007.

[11] A. Robins, J. Rountree, and N. Rountree. Learning and teaching programming: A review and discussion. *Computer Science Education*, 13:137–172, 2003.

[12] H. Roumani. Design guidelines for the lab component of objects-first cs1. In *SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education*, pages 222–226. ACM, 2002.

[13] J. C. Spohrer and E. Soloway. Novice mistakes: are the folk wisdoms correct? *Commun. ACM*, 29(7):624–632, 1986.

[14] A. Vihavainen, M. Paksula, and M. Luukkainen. Extreme apprenticeship method in teaching programming for beginners. In *SIGCSE '11: Proceedings of the 42nd SIGCSE technical symposium on Computer science education*, 2011.

[15] L. S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, Cambridge, MA, 1978.

[16] L. Winslow. Programming psychology - a psychological overview. *SIGCSE Bulletin*, 27:17–22, 1996.