Three Years of Design-based Research to Reform a Software Engineering Curriculum

Matti Luukkainen, Arto Vihavainen, Thomas Vikberg
University of Helsinki
Department of Computer Science
P.O. Box 68 (Gustaf Hällströmin katu 2b)
Fi-00014 University of Helsinki
{ mluukkai, avihavai, tvikberg }@cs.helsinki.fi

Final draft

Originally appeared as: M. Luukkainen, A. Vihavainen and T. Vikberg. Three years of design-based research to reform a software engineering curriculum. In Proceedings of the SIGITE '12. ACM, 2012.

Abstract

Most of the research-oriented computer science departments provide software engineering education. Providing up-to-date software engineering education can be problematic, as practises used in modern software development companies have been developed in the industry and as such do not often reach teachers in university contexts. The danger, and often the unfortunate reality, is that institutions giving education in software engineering end up teaching the subject using outdated practices with technologies no longer in use. In this article we describe a three-year design-based research where the goal has been to design and reform a software engineering subtrack within our bachelor curriculum that would make it possible for the students to have strong up-to-date theoretical and practical skills in software engineering without a need to remove any of the existing theoretical aspects.

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

General Terms
Design, Experimentation, Management

Keywords instructional design, modern software engineering practises, computer science curriculum

1 Introduction

Bachelor level computer science (CS) education has many goals. A CS curriculum should offer students a solid theoretical foundation in CS as well as convey enough practical skills and knowledge on current technologies. Although the focus of many CS departments in research universities lies in theoretical fields, such as algorithmics and computational complexity, most universities embed software engineering (SE) studies in their curriculum. This poses a great challenge for SE educators as they need to keep up with the rapid development of the IT industry. The danger, and often the unfortunate reality, is that institutions giving SE education end up teaching SE using outdated practices with technologies no longer in use [1].

According to IEEE Software Engineering is *(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)* [3]. In contrast to more established fields of CS, the field of SE is still rather immature and has seen a radical paradigmatic change in the past decade with the emergence of agile methodologies [7, 8, 34]. In contrast to traditional plan-based methods (e.g. the so-called Waterfall method [32]), agile methodologies emphasize working software, customer collaboration, human interaction and best technological practices [8].

It has lately been recognized that SE is more a craft than a science [30, 31] and as such it is to be treated differently than the more theory-based topics. The up-to-date industrial best practices and tools should be taken into account when teaching SE. It has been noticed that with a traditional university instruction students even need to unlearn working methods acquired in formal education when entering the industry [9, 23].

One of the major problems is that most university teachers lack recent industrial experience or direct contact to industry, and as such are not aware of the emergent modern SE practises such as agile methods [1]. As teachers have not used them, they are not truly capable of conveying agile methodologies in teaching as agile methodologies can often only be learned through practicing them [34].

These issues have led to a situation where the SE-related bachelor level education is heavily out of synch with reality. Teaching yesterday's theories poses problems for the credibility of the education and leads to a lack of motivation amongst students with existing industrial experience.

The main SE learning objective of a graduating bachelor at our department is that the student should be able to perform efficiently in a modern software development project. In this paper we describe a three-year design-based research where the goal has been to design a SE subtrack within our CS bachelor curriculum that would make it possible to have strong up-to-date theoretical and practical skills in SE.

2 Initial problem analysis

The aim of design-based research [35, 10] is to construct an educational artefact [12, 26], in this case a reformed SE subtrack within our CS curriculum. The artefact is an intervention into current practices [11] and thus the process of developing the artefact begins by initial problem analysis [22]. The development of the artifact is then continued in an iterative manner where the evaluation process is undertaken repeatedly at the beginning of each iteration as well as during iterations [13, 26]. This should lead to an improved artefact and improved understanding of the process that led to the improvement [16, 33]. Next we describe the situation of the SE subtrack of the Department of Computer Science at the University of Helsinki at the beginning of the first iteration that was started in fall 2009.

The SE subtrack was based on the courses displayed in Figure 1. During the first semester the students undertake the courses Introduction to programming, Advanced Programming (together forming CS1) and Software Modeling. In the second semester they take the course Data Structures (CS2), Programming Project, and Introduction to Databases. In their second year they carry out the Database Application Project and our SE course. During the third year the students undertake what could be seen as the culmination of their SE studies so far: SE Project, a 14-week capstone project done in groups of 4-6 students. The capstone project requires roughly 200 working hours for each participant. On top of the SE subtrack the students take other courses with focus on algorithmics and networking, as well as minor-subject studies.

Evaluation of the effectiveness of SE subtrack was done by observing **(1)** the capability of students to work in the Software Engineering Project **(2)** the progress they made during the SE Project **(3)** their readiness to start working as a "junior software developer" in the software engineering industry after the project was finished.

Observations for (1) and (2) were made during the SE Project course by project instructors. Observations for (3) was done in the Software Factory, a new master's level advanced software project [2]. Those of the students who are continuing beyond bachelor and are majoring in SE take part in the Software Factory which is a seven-week, five days a week project. It is done for a real client within self-organizing teams which are assumed to follow all the best practices of agile software development.

Students participation in Software Factory gave us a fine opportunity for evaluating their performance as junior software developers. The evaluation was done by the coordinator of the Software Factory project, who is an expert in industrial practices, and by a senior faculty member, both members of the curriculum design team.

Several problems were identified during the observations:

(P1) the programming skills and routine of most of the students were not at an adequate level

(P2) good programming practices [30, 25] were missing leading to lack of maintainability of the written code

(P3) no knowledge of theories or practices of testing
(P4) no practical knowledge of software design and architectures
(P5) lack of experience in designing and programming web applications
(P6) outdated knowledge of software development processes
(P7) lack of elementary SE tool skills and knowledge such as version control, integrated development environments (IDEs), shell scripting and system administration
(P8) complete unawareness of agile SE practices such as user-story-based requirements engineering, planning and estimation [17] and more technical practices such as test-driven development [6], continuous integration [21] and behavior-driven development [15]
(P9) lack of team working skills and practises

3 The reform strategy

The goal of the reform was to raise the standard of the SE Project and to tackle the identified problems listed in the previous section within some of the courses without damaging the course passing rates. The key course for preparing students for the SE Project was the SE course that is typically taken at the later part of the 2nd year, just before attending the SE Project. However, the design

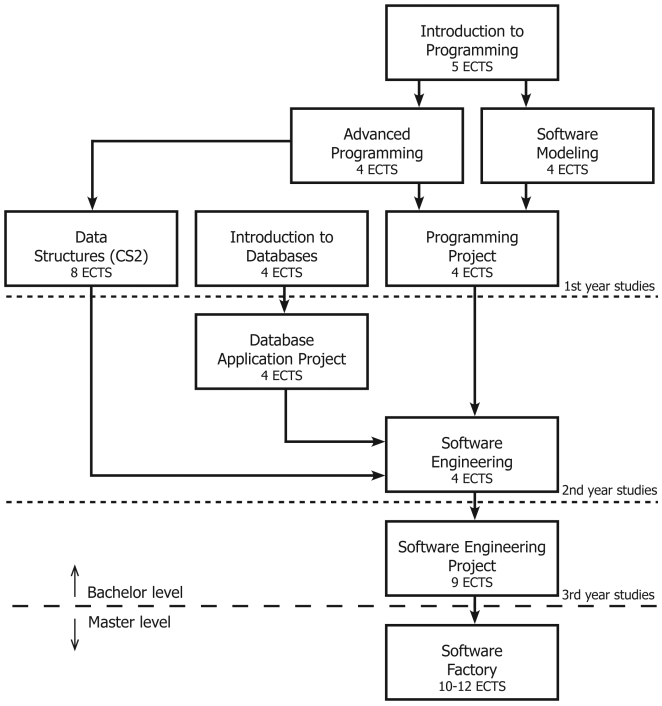

Figure 1: SE subtrack at University of Helsinki.

team felt that reforming a six-week 4 ECTS[1] course would not be enough to root the required theoretical and practical background in students. Thus, it was decided that each of the courses leading to the SE course had to be reformed.

When this goal was set, a typical implementation strategy would have been to reform the course structure within the curriculum. This approach is both bureaucratic and time consuming, as the changes have to be ratified by different instances, after which they can only be put into action after a certain time[2]. This approach is also inefficient as it often leads to introduction of courses into specific knowledge areas, which in turn would not affect other courses. For example creating a course in software testing would not force good testing practices into other courses.

Instead of introducing new courses the reform strategy chosen was to completely rethink the contents, structures and pedagogical practices of the existing courses. It was decided that up-to-date industrial best practices should be introduced from day one and emphasized even in more theory-oriented courses such as CS2. This emphasis on practical undertakings is a major shift from the typical approach taken before at our department where the goal was to teach the *idea* of software development on a high level.

The departmental competencies lie in the theory of SE and not in the industry practices. This forced the design team to employ an unusual strategy for a research university for laying out the reformed curriculum. The main responsible faculty member of the reform – tenured assoc. prof. – went on to participate in the Software Factory as a normal student to find out the problems himself and to engage in a real industry project. The development team also recruited students within the department with strong industry backgrounds as developers and teachers for the reformed courses.

4 Design iterations

The first design iteration took place in fall 2009 and has since progressed into a completely reformed SE subtrack in spring 2012. During the three-year period one or two courses in each iteration were restructured. Each iteration lasted one academic term when the courses reformed were given. In each iteration, courses already reformed were further refined according to experiences and data gathered from the students' performance in subsequent courses. Next, we will present an overview of the iterations.

Fall 2009

The reform took a rather cautious start. The first step was to integrate the Software Modeling course to progress lockstep with the Advanced Programming course which was taken at the same time by the same set of students. The

[1] European Credit Transfer and Accumulation System credits where one academic year corresponds to 60 ECTS.

[2] The course structure of the curriculum can be changed at the University of Helsinki in two-year intervals.

content of the course (UML class and sequence diagrams and introduction to object design) remained much the same as it had been before. The course was set to emphasize iterative and agile software processes in contrast to the more traditional approaches emphasized before. Also, the UML-modeling was done in a less rigorous manner than before with the aim to introduce agile modeling principles [5], where UML is used as a help for communication and design during the program development phase, not as a comprehensive documentation tool. Thus, during the first term only item (P3) from the problems listed in Section 2 was addressed.

Spring 2010

The design team felt that the main problem was item (P1): the lack of adequate programming skills that the students possessed even after several years of study. The team decided that the old lecture-based programming instruction had to be replaced by a more pragmatic approach with a heavy emphasis on actual programming under guidance of more experienced programmers.

As a result a new pedagogical model was developed for introductory programming courses, the *Extreme Apprenticeship* method (XA) that is based on principles of Cognitive Apprenticeship [19, 18] as well as modern SE practises.

XA was highly successful from the point of view of learning outcomes and was also well received by the students [36]. The new method of programming instruction boosted the programming skills of our students considerably. Programming used to be a disliked topic among a remarkably large portion of the student population. This made the further development of our SE subtrack much easier.

Summer 2010

As a continuation to XA the Programming Project course was arranged in a completely new form. The project consists of students designing and implementing a program on their own: typically a game such as Tetris or a simple data organizing application.

The project used to be organized to imitate the Waterfall-development model: the first three weeks contained analysis and design, which was followed by two weeks of programming ending in a single week of testing. Heavy documentation was required at the end of the project. Usually the students' working process was reduced to a "code and fix"-type of programming effort where design and testing were almost completely ignored, and the resulting documentation was a mix of initial plans and a final document that was often reverse engineered from the actual implementation.

In order to work on the problem items (P3), (P6) and (P8), the course was reformed into an agile, iterative project, where each iteration lasted a week during which a small increment of the program was to be completed. After six iterations the program should be completed. Along the implementation the students were required to build automated unit tests using the JUnit-testing framework [6], and also to have up-to-date UML-based implementation documentation sketches at all times.

Fall 2010

The XA-model of programming instruction was further elaborated and scaled to a much larger audience than the first instance [37]. More emphasis was put on code quality (P2), and not just the correct behavior of the code.

The Software Modeling course was aligned with the XA type of programming instruction. As automated tests were added as a mandatory part of the Programming Project, there was a need to incorporate them into the learning objectives of at least one of the previous courses as well. Automated unit testing was incorporated as a part of the software modeling course, leading to further work on the problem item (P3).

Spring 2011

XA-based programming instruction was further elaborated by completely removing the lectures altogether (lectures had previously been decreased from four weekly hours to two) and focusing even more on the students' active problem solving [28].

In order to work on the problem item (P7), the Programming Project was iterated to enforce basic use of a version control system (VCS), that had not previously been emphasized. This led to an extremely inefficient workflow in the SE Project. Since the programming project is a one-person project, the students had a relaxed environment for learning the basics of a VCS workflow without more challenging VCS-related issues such as merge conflicts.

Problem items (P1) and (P2) were focused on in a significant change within the Data Structures course [29]. The Data Structures course is a typical CS2-course covering basic data structures (stack, binary search tree, balanced binary search tree, heap, graph) and the related algorithms. The old approach was based on mathematical analysis and pseudo-code, no real programming was done in the course. The course had a rather high drop-out rate and most of the students passing the course were not particularly good in applying the theory to practical problem solving.

The design team decided to refocus the CS2 course to contain more practical problem solving tasks that were implemented using a real programming language instead of pseudocode. A total of sixty new programming tasks were included into the course, some of which were quite challenging. All theoretical aspects were also kept in the course, which led to an increase in the workload of students in contrast to the old CS2 course.

Part of the lectures were restructured to contain worked example-style [14] problem solving, and best programming practices were emphasized throughout the course. Increased retention rate, observed learning outcomes, and positive student feedback showed clearly that the restructuring of the course was a success [29].

Fall 2011

The Software Modeling course saw its third refinement in fall 2011[3]. The new

[3]Because of the shifts in content the course was renamed to "Methods for Software Engineering" starting from fall 2012.

course put even more emphasis on agile processes and agile modeling principles that center around maximal enhancement of communication among software developers and other stakeholders.

The course introduced communication and team-related aspects in the form of design and modeling sessions. In the sessions students worked in teams of 4-6 people around a large whiteboard designing and modeling larger tasks that would have been very challenging for most of the students to be done on their own. Whiteboards were used to emphasize that designs can always be changed and improved. The amount of lectures was also reduced, which allowed focusing more on active problem solving during the course. These modifications addressed items (P4), (P6), (P8) and (P9).

Spring 2012

In spring 2012, major changes were made to improve the SE curriculum. A new course focusing on web application development was introduced and the first implementation of a totally reformed SE course took place.

Some web application development had been done as part of the curriculum already before the new course. The Database Application Project that had been in the SE curriculum for more than a decade had slowly evolved to a project where the students designed and implemented a web application that was backed by a database. Most students learned some PHP or Java Servlet techniques on their own in order to implement the application. Before XA-style programming instruction, programming skills were poor and there was a total disrespect for code quality and structure. The web applications that students created during their project did not usually follow any of the best practices of web programming, such as separation of representation, structure and behavior or layered architecture [4, 27].

To remedy this problem (P5), the design team set up the Web Application Development course, where students were guided to do web programming *the right way*. Strong emphasis was put especially on the maintainability of written code, further addressing the problem item (P2). The course was scheduled to be taken in the second term of the second year, just before the SE course. In contrast to the rest of the SE subtrack, Web Application Development was not compulsory. However, roughly half of the students took the course.

The design team completely restructured the SE course for the first time in the later part of spring 2012, just after the new Web Application Development course. The previous SE course was a theoretical walkthrough of phases found in any software development methodology: requirements analysis, design, testing and software maintenance. The course focus was on the Waterfall model and its interpretation of the role of the phases in the development process. This view was far too theoretical, abstract and outdated from the point of view of current industrial practises that rely heavily on agile development [8, 34, 7].

The reforms made to earlier courses in the study path had given a partial remedy to many of the problems listed in Section 2. The rest needed to be taken care of in the SE course. The learning objectives of the course were set high in order to provide students with the capability of working productively in the SE

Project that was arranged as an intensive Scrum-like agile project. The specific learning objectives were:

- learn the basics of Scrum
  - iterative and incremental development
  - roles: Scrum Master, Product Owner, Developer
  - meetings: sprint planning, daily scrum, review, retrospective
  - artifacts: product and sprint backlog
- agile planning, requirements engineering and execution of an iteration
  - user stories, their estimation and prioritization
  - product backlog grooming
  - planning game to maximize the customer value of the developed product
  - splitting stories to tasks in sprint backlog
  - reporting the progress within an iteration
- concepts of program testing and practical techniques in automatic testing
  - test-driven development
  - story level testing, behavior-driven development
  - continuous integration
- support for program design
  - layered architecture and domain-driven design
  - object-oriented principles: encapsulation, cohesion, low coupling, testability, maintainability
  - design patterns
- efficient use of Git version control system in group work
- use of static analysis tools to give diagnostics on developed programs

The new course implementation consisted of three components. Twenty hours of lectures contained a broad introduction to the covered topics. Weekly task sets aimed at building routine in the use of the Git version control system, design patterns [24] and the use of the Spring framework [38]. Both the lectures and the tasks set the stage for the third component: a SE mini-project. The mini-project was a simulation of an agile SE project done in three iterations, each taking one week. The mini-project was done in groups of 3-5 students. Each group (22 altogether) was given the same topic and the course instructors acted as customer, product owner and scrum master. The developed program was not particularly large, and the focus was on gaining more in-depth knowledge of agile principles and practices that were introduced to students in the lectures.

The working hours for each student within an iteration were limited to four hours. This was an important aspect since one of the most critical skills in agile development is the capability to commit to a workload (e.g. a set of *user stories*) that the team believes can be implemented in high quality (including automated tests) within one iteration [8, 34]. Committing to implement a certain set of user stories in an iteration also motivates students to estimate the work that is required for implementing each user story. The user story management was done with *product* and *sprint backlogs* as the current industry best practices recommend [17]. The project groups were encouraged to have proper sprint planning sessions and retrospectives during the mini-project.

Students found the mini-project a beneficial experience. Most of the groups struggled during first iteration but thanks to the inspect-and-adapt nature of iterative development and coaching given by the course instructors, all groups matured substantially during the three iterations. During the mini-project, students saw an immediate benefit of the theories from the lectures and various tools and techniques practised in the normal exercises.

The design team evidenced a similar phenomenon in the SE course as in the CS2 course a year earlier: when compared to the traditional lecture-based course the students learned the theoretical topics at a deeper level when the course consisted of active student participation and practical tasks.

5 Analysis and further work

After three years of iterations the SE curriculum has almost been renewed. The quality and the working process in the currently ongoing SE Project course shows that the students have progressed towards the goal set for their SE subtrack. They are much more prepared to participate in software projects using modern agile practices and use up-to-date methods and tools when compared to the students that took the pre-reform SE subtrack.

One of the main objectives was to gradually introduce various practices and theoretical ideas to the courses. This has resulted in strong inter-course links, introducing a concept in one course and the development of the same theme further in another course. E.g. the idea of automated testing together with some practical tasks are introduced in the course Software Modeling. When students take part in the Programming Project, they are required to have a rather comprehensive unit testing harness. Finally, in the SE course, the theories and concepts behind testing are covered more thoroughly and agile practises such as TDD and BDD are introduced to students within a realistic setting. This is a big difference to the old model where one concept was taught mostly only within a single course and the connections between courses were rather weak.

During the new course Web Application Development we made an important finding: despite the much increased programming routine, a considerable amount of students struggled in sysadmin-like issues such as working in command line, shell scripting, and server installations. These issues are important in web application development since a web application also involves the server components. The weak sysadmin skills were surprising. It had been assumed for a long time that sysadmin-related skills are easily learned by each student on their own and having those as part of university courses would not be necessary or even wise. However, it has been recognized that nowadays the clear separation of software developers and administrators is no longer valid in the software industry, at least in smaller companies [20]. Thus some level of sysadmin skills must be included in the SE curriculum.

We have embedded some sysadmin skills already in the new SE course (more emphasis on working with command line with VCS and Build management

systems such as Maven). In the future the introduction of sysadmin concepts and skills should be embedded in several courses, as is currently done with testing. Suitable courses where the existing theory could easily be enriched with these topics are Operating Systems (working with command line, shell scripting), Introduction to Databases (installation and operation of a database server), Introduction to Computer Networks (internet-related command line tools such as telnet, traceroute and curl, installation and operation of a web server).

The courses Introduction to Databases and Database Application Project are currently in a rather old-fashioned state. The courses have remained almost intact for more than 10 years and are heavily based on relational algebra and relational database model. A reform is clearly needed: active student participation should be increased, the old dust should be wiped out. Also, strong connections to the programming courses should be established. Currently the courses do not at all mention the important concept of object relational mapping, i.e, how objects in a program implemented e.g. in Java could be persisted to a relational database. Further, the emergence of nosql-databases should be taken into account in the curriculum. These improvements are in focus in the next iterations of the curriculum reform.

6 Conclusions

In this article we have presented three years of design-based research to reform our software engineering curriculum. During the reform we have made no compromises. None of the core CS topics have been dropped, while relevant content and new approaches have been introduced.

The renewed versions of CS2 and SE courses contain all the old theoretical elements as well as a substantial amount of practical problem solving. The average student workload has increased considerably. Although the average workload has increased heavily, the student feedback has been overwhelmingly positive. The students seem to enjoy working more as the connections to the reality of the software industry are made clear.

How is this possible? We have a strong feeling that the reform would not have been possible without XA-based programming instruction that has raised the programming routine of our students to a completely new level as well as introduced them to working hard starting from day one. The reformed CS2 and SE can be seen as a direct continuation of active learning-based education, which boosts student motivation and confidence and by that helps students to grasp also the more theoretical topics more easily.

The reform process has been research driven [22] with publications linking current scientific research to various design iterations. Our work has not only produced a renewed SE curriculum, but also new student-centered teaching methods [36, 37], renewed ways of presenting old content [29] and new ways of organizing administration [28].

References

[1] P. Abrahamsson. Inaugural lecture: The new era of software quality (in Finnish). http://www.cs.helsinki.fi/kolumni/ohjelmistojen-laadun-uusi-aikakausi, December 2009.

[2] P. Abrahamsson, P. Kettunen, and F. Fagerholm. The set-up of a valuable software engineering research infrastructure of the 2010s. In *Workshop on Valuable Software Products*, 2010.

[3] A. Abram, J. W. More, B. Pierre, and D. Robert, editors. *Guide to the Software Engineering Body of Knownledge*. IEEE Computer Society, 2004.

[4] D. Alur, D. Malkis, and J. Crupi. *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall, 2 edition, 2003.

[5] S. W. Ambler and R. Jeffries. *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*. Wiley, 2002.

[6] K. Beck. *Test Driven Development: By Example*. Addison-Wesley, 2002.

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

[8] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mallor, K. Shwaber, and J. Sutherland. The Agile Manifesto. Technical report, The Agile Alliance, 2001.

[9] A. Begel and B. Simon. Struggles of new college graduates in their first software development job. In *Proceedings of the SIGCSE '08*. ACM, 2008.

[10] P. Bell. On the theoretical breadth of design-based research in education. *Educational Psychologist*, 39(4), 2004.

[11] C. Bereiter. Design research for sustained innovation. *Cognitive Studies Bulletin of the Japanese Cognitive Science Society*, 9(3):321–327, 2002.

[12] A. L. Brown. Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *The Journal of the Learning Sciences*, 2(2):141–178, 1992.

[13] H. Burkhardt and A. H. Schoenfeld. Improving educational research: Toward a more useful, more influential, and better-funded enterprise. *Educational Researcher*, 32(9):3–14, 2003.

[14] M. E. Caspersen and J. Bennedsen. Instructional design of a programming course: a learning theoretic approach. In *ICER '07: Proc. of the 3rd international workshop on Computing education research*, 2007.

[15] D. Chelimsky, D. Astels, B. Helmkamp, D. North, Z. Dennis, and A. Hellesoy. *The RSpec Book*. Pragmatic Bookshelf, 2010.

[16] P. Cobb, J. Confrey, A. diSessa, R. Lehrer, and L. Schauble. Design experiments in educational research. *Educational Researcher*, 32(1):9–13, 2003.

[17] M. Cohn. *Agile Estimating and Planning*. Prentice Hall, 2005.

[18] A. Collins, J. Brown, and A. Holum. Cognitive apprenticeship: Making thinking visible. *American Educator*, 15(3):6–46, 1991.

[19] A. Collins, J. Brown, and S. Newman. Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In L. Resnick, editor, *Knowing learning and instruction Essays in honor of Robert Glaser*, volume Knowing, l of *Psychology of Education and Instruction Series*, pages 453–494. Lawrence Erlbaum Associates, 1989.

[20] Devops: A software revolution in the making? *Cutter IT Journal*, 24(8), 2011. Special issue.

[21] P. Duval, S. Matyas, and A. Glower. *Continuous Integration: Improving Software Quality and Reducing Risk*. Addison-Wesley, 2007.

[22] D. C. Edelson. Design research: What we learn when we engage in design. *The Journal of the Learning Sciences*, 11(1):105–121, 2002.

[23] A. Fox and D. Patterson. Crossing the software education chasm. *Communications of ACM*, 55(5):44–49, May 2012.

[24] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[25] A. Hunt and D. Thomas. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley, 1999.

[26] K. Juuti and J. Lavonen. Design-based research in science education: One step towards methodology. *NorDiNa*, 4:54–68, 2006.

[27] J. Keith. *DOM Scripting: Web Design with JavaScript and the Document Object Model*. friends of ED, 2005.

[28] J. Kurhila and A. Vihavainen. Management, structures and tools to scale up personal advising in large programming courses. In *Proceedings of the SIGITE '11*. ACM, 2011.

[29] M. Luukkainen, A. Vihavainen, and T. Vikberg. A software craftsman's approach to data structures. In *Proceedings of the SIGCSE '12*. ACM, 2012.

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

[31] P. McBreen. *Software Craftsmanship: The New Imperative*. Prentice Hall, 2001.

[32] W. Royce. Managing the development of large software systems. In *Proceedings of IEEE WESCON 26*. TeX Users Group, August 1970.

[33] A. H. Schoenfeld. Bridging the cultures of educational research and design. *Educational Designer*, 1(2), 2009.

[34] K. Schwaber and M. Beedle. *Agile Software Development with SCRUM*. Prentice Hall, 2002.

[35] The Design-Based Research Collective. Design-based research: An emerging paradigm for educational inquiry. *Educational Researcher*, 32(1):5–8, 2003.

[36] A. Vihavainen, M. Paksula, and M. Luukkainen. Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the SIGCSE '11*. ACM, 2011.

[37] A. Vihavainen, M. Paksula, M. Luukkainen, and J. Kurhila. Extreme apprenticeship method: key practices and upward scalability. In *Proceedings of the ITiCSE '11*. ACM, 2011.

[38] C. Walls. *Spring in Action*. Manning Publ., 2011.