Scaffolding Students' Learning using Test My Code

Arto Vihavainen, Thomas Vikberg, Matti Luukkainen, Martin Pärtel

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

Final draft
Originally appeared as:
Vihavainen, A., Vikberg, T., Luukkainen, M., and Pärtel, M. (2013). Scaffolding students' learning using Test My Code. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, ITiCSE 13, pages 117–122 , New York, NY, USA. ACM.

Abstract

As programming is the basis of many CS courses, meaningful activities in supporting students on their journey towards being better programmers is a matter of utmost importance. Programming is not only about learning simple syntax constructs and their applications, but about honing practical problem-solving skills in meaningful contexts. In this article, we describe our current work on an automated assessment system called Test My Code (TMC), which is one of the feedback and support mechanisms that we use in our programming courses. TMC is an assessment service that (1) enables building of scaffolding into programming exercises; (2) retrieves and updates tasks into the students' programming environment as students work on them, and (3) causes no additional overhead to students' programming process. Instructors benefit from TMC as it can be used to perform code reviews, and collect and send feedback even on fully on-line courses.

Category[K.3.2]: **Computers and Education** Computer and Information Science Education
Terms: Experimentation, Human Factors, Measurement
Keywords: programming, testing, automatic assessment, verification, extreme apprenticeship, situated learning

1 Introduction

We organize our programming courses using the Extreme Apprenticeship method (XA) [16]. One of the success factors in XA is that all learning-oriented activities are as "genuine" as possible, e.g. programming is done using industry strength tools while honing good programming practices. The work that students are required to do must be well defined and achievable with students' existing knowledge, especially in the early phases of a course. Early success allows students to build both self-confidence and their programming routine, which helps them to transition towards seeing more than simple syntax.

Success in the early parts of the courses relies heavily on small and relatively easy exercises, which are designed to help understand course-specific learning objectives. For example, during the first week of our CS1 course, the students practice basic input and output, and build programs that use conditionals with almost 30 different exercises [1]. As students progress in the course, the small exercises combine into bigger programs. This is to demonstrate and teach how larger programs are solved in a step-wise manner.

The students work on the exercises in computer labs where their learning is scaffolded by numerous teaching assistants (TAs) [17]. Scaffolding means providing well-timed support to the learners' learning process, so that they can achieve such learning objectives that they could not reach on their own [18]. In addition to scaffolding, the instructors assess exercises, and provide feedback on student's programming practices and program design.

Although scaffolding and bi-directional feedback is usually beneficial for both the student and the instructor, parts of scaffolding tends to be repetitive. As there are tens of students in the labs at the same time, it is hard to build a larger picture of a students' progress.

Test My Code (TMC) has been developed to reduce the amount of repetitive tasks of the TAs, i.e. exercise checking and some parts of the scaffolding, and to increase the amount of time instructors have for mentoring and supporting students. TMC also supports gathering snapshots from the students' programming processes, as well as providing long-distance support via code reviews. It can be useful for instructors and CS education researchers alike, as one can choose how it is to be utilized in a class; it can be used as an assessment service and a data gathering tool.

This paper is organized as follows. In section 2, we motivate the work on a new assessment system, after which we give a description of programming exercises in an XA context. Section 4 describes the structure of TMC, after which we describe how it can be used to scaffold students' learning. Finally, we give an overview of the performed evaluations, the impact of TMC on our courses, and lay down planned future work.

[1] See Object-Oriented Programming with Java at `http://mooc.fi`.

2 Motivation for a new System

Several of the programming courses at the University of Helsinki are organized using the Extreme Apprenticeship (XA) method, which is a modern interpretation of apprenticeship-based learning [16]. XA emphasizes students' individual effort and communication between the learner and the advisor. Core values in XA include:

- A craft can only be mastered by actually practicing it, for as long as is necessary. To be able to practice the craft, students need meaningful activities, i.e. exercises.
- Continuous feedback between the learner and the advisor. The learner needs confirmation that she is progressing *and* in a desired direction. Therefore, the advisor must be aware of the successes and challenges of the learner throughout the course.

In order for XA to properly benefit from an automatic assessment system, the system has to accommodate scaffolding as well as support the situative perspective on learning in apprenticeship-based education [3]. The perspective views the situations where knowledge is developed and later applied as highly connected, as "methods of instruction are not only instruments for acquiring skills; they also are practices in which students learn to participate" [9].

We want our students to become proficient in programming, not only to know about programming. According to the situative perspective, abstracting theory from practice does not yield transferability [5]. As a goal of our CS education is to help students on their journey towards becoming experts in their field, chosen tools and methods have to allow "participation in valued social practices" [9] of the respective professional communities. For aspiring programmers, the tools and *practices of learning* [9] in their training should be similar to those used in the software engineering industry. Industry best practices, such as code reviewing [8], has to implemented in a way that enable instructors to perform the tasks in a non-intrusive manner.

Scaffolding of students' learning has to be well-timed and not over excessive. Due to the nature of XA, accumulated knowledge of the learning of students process is continuously gathered through the bi-directional feedback of the courses. The ability to transform this knowledge iteratively into the exercises and tests of the courses has to be featured in the automated assessment system.

There exist lots of automatic systems that are designed to assess students' programs [6, 1, 11]. However, as pointed out by Ihantola et al. [11], most of them are created for a specific course or as a part of a research project, and are not made available for distribution or modification. Two of the exceptions are the Marmoset project [14], and the Web-Cat project [7], both of which are available as open source projects.

Most of the currently available assessment systems are web-based, which means that in order to solve an exercise, a student has to download a template from a web-page, solve it, and then submit it using a web-GUI. If the tests in the assessment system are built to provide feedback to the student, retrieving the feedback from a web-page causes an extra step. In addition to poorly supporting the learning of specific tools and practices, the use of a web-GUI for downloading and submitting each would cause lots of unnecessary overhead. Constantly zipping and unzipping of projects is also not something we wish to teach our students as it does not belong to the workflow of a professional software engineer.

2.1 Requirements for Test My Code

Almost all CS and IT curricula contain courses on web-development and algorithmics, both of which benefit from tools that can be utilized to support students during their learning process.

In order to create a realistic web-development environment, the students should be able to e.g. configure downloading of dependencies and deploy the same application both locally and online. Local deployment is useful e.g. as students practicing web development should also learn to debug web applications using tools such as integrated developer consoles in web browsers. In order to properly assess and scaffold good development practices, support for testing both frontend and backend functionality is of great importance.

In algorithmics courses the learning objectives are usually a mix of analysis of run-time complexity, and the creation of implementations. Many of the current assessment systems handle algorithm run-time analysis using naive run-time clocking. Although this is sufficient for most of the cases, our large-scale courses have peaks e.g. during deadlines, which can cause false negatives in the tests. Additional false negatives are caused due to the use of cloud-based virtual machines that have fluctuations in the available processing power. One approach for handling this is deterministic algorithm analysis using bytecode counting, see e.g. [12].

As an ongoing effort, Test My Code (TMC) is currently designed to

- **be as minimally intrusive as possible**; the assessment service does not introduce any additional overhead to the students' working process. TMC downloads the exercises directly to the working environment, and supports both local and server-side tests
- **support timely scaffolding within the exercises**; new goals can be made visible only after previous ones have been finished, and adding scaffolding messages to the tests is easy
- **allow awarding points for completing smaller goals**, not just for completing full exercises
- **support bidirectional feedback**; TMC supports code reviews as well as direct feedback regarding the exercises both from the students' and the instructors' perspective
- **make testing visible**; the actual testing process is made visible, which eases students into the thought of writing their own tests

- **support web-application development courses**; testing of both front-end and backend functionalities, as well as ad-hoc downloading of dependencies, is made possible using Maven[2]
- **support algorithm testing** in unstable environments

From the course instructor's perspective, TMC

- **causes no additional overhead from the management perspective**; the system has a clean integration interface for reading in assessed exercises
- **allows mistakes in the exercise generation process**; if an exercise contains mistakes, updated versions can be easily published to students
- **allows honing software engineering practices for exercise developers**; generating exercises and tests does not substantially differ from work done in a normal software engineering context
- **gathers data from students' programming process** for future analysis and e.g. plagiarism detection

In addition, TMC is open source and is freely available[3].

3 Test My Code

In its current state, TMC consists of several components that are organized using client-server architecture, see Figure 1. The NetBeans plugin

- retrieves and updates course exercises from an assessment server
- displays built-in scaffolding messages during the working process
- submits exercises to the assessment server
- allows giving and receiving direct feedback regarding the exercises
- gathers data from students' programming process

In addition, the plugin introduces a new menu option called TMC and three new toolbar buttons. The TMC menu contains options for changing settings (e.g. username, course, exercise directory), checking for new exercises, submitting answers, and requesting and viewing code reviews. The toolbar buttons are added for (1) running the currently modified application independently of any accidentally selected main project, (2) running local tests, and (3) submitting the solution TO the the assessment server. If the student presses the "run

[2]`http://maven.apache.org`
[3]`http://github.com/testmycode`

tests locally" button, locally available tests for the exercise are run and possible scaffolding messages are shown immediately.

The web-interface allows students to register to a course, view their statistics, and optionally submit exercises outside the IDE. Course instructors use the web interface for administrative tasks such as the creation of new courses, refreshing exercises, viewing submissions, responding to code review requests, and viewing course-related statistics.

Requests to the TMC backend are routed using one or more web servers. Each course has a git-repository that contains the course exercises. Students' submissions and information are entered into a database. Once a student submits an exercise, the exercise is verified on one of the sandbox servers that run transient user-mode Linux[4] virtual machines. Each sandbox server has an optional Maven cache for storing library dependencies for e.g. web development-related exercises.

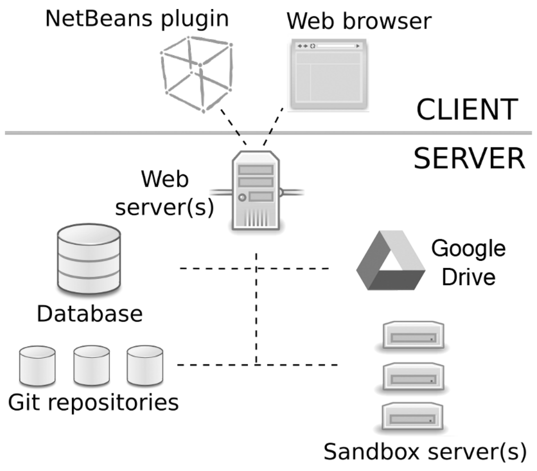


Figure 1: TMC architecture

3.1 Code Reviews

Code reviews are designed to identify defects and point out improvement opportunities in the reviewed software [8]. Depending on the programming language used, some of the code reviewing can be automated. For Java typical tools are e.g. Findbugs [10] and Checkstyle, and even they fail to point out improvement opportunities in the program design and architecture. Hence, manual review is still often useful.

TMC provides code review capability in the web interface, see Figure 2. Once the students have submitted their projects, the instructors can browse the submitted code online. If students request a review for their code, the requests

[4]`http://user-mode-linux.sourceforge.net/`

are visible on the TMC main page. Problems identified during a review process are communicated to the developer, who then further works on the issues.

Reviews can either be requested manually in the IDE, or the instructors can spontaneously review students' code. Once a review has been performed, the student is notified via email or directly within the IDE. As the student opens the IDE (or if the IDE is running), she sees the review comments immediately.

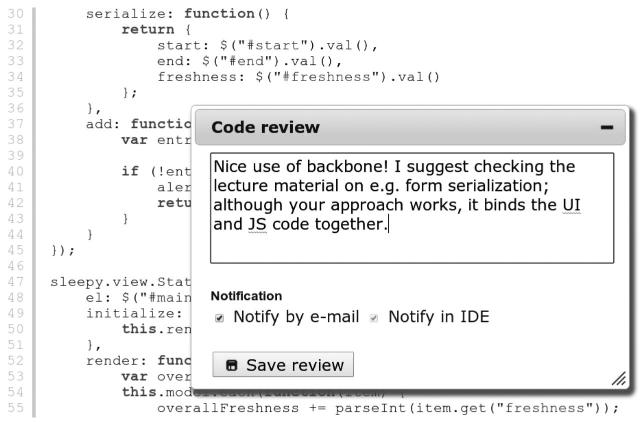
In our programming courses, in addition to the scaffolding in XA labs, we often perform manual code reviews at least for open exercises that do not impose a specific structure for the program. A single code review usually takes between 5 and 10 minutes, and each student typically receives at least one weekly review (assuming she has programmed during that week). Having manual code reviews is possible due to the way we organize our instruction [13].

3.2 Creating Exercises and Tests

Creating a new exercise and its tests starts with creating a programming project (e.g. a standard Maven project), and continues by working on the exercise using the same steps that one would while using TDD [2]. For each part of the exercise, one first creates the tests and error messages that indicate what went wrong, and afterwards add the functionality that makes the tests pass.

Once the exercise is finished, further work is needed. The finished exercise files are used as both the model solutions and the template that the students receive from TMC. The content in the version that students receive needs to be altered. Altering content is done based on comments within the project files. For example, a source file that starts with a comment `// SOLUTION FILE` is never sent to the student.

After altering the source code files is finished, the tests usually need modification. The modified version does not usually contain all the files that are referenced by the original tests. Depending on the programming language used,



```
    serialize: function() {
        return {
            start: $("#start").val(),
            end: $("#end").val(),
            freshness: $("#freshness").val()
        };
    },
    add: functio
        var entr

        if (!ent
            aler
            retu
        }
    }
});

sleepy.view.Stat
    el: $("#main
    initialize:
        this.ren
    },
    render: func
        var over
        this.mode
            overallFreshness += parseInt(item.get("freshness"));
```

Nice use of backbone! I suggest checking the lecture material on e.g. form serialization; although your approach works, it binds the UI and JS code together.

Figure 2: Reviewing students' code (the input window is resizeable).

the tests need to be modified to tolerate conditions such as missing classes and methods. TMC provides a Java DSL that allows convenient access to student code via reflection, and provides user-friendly scaffolding messages in common error situations.

Once the tests are finished, points that the students receive for the exercise are added. Points for an exercise or a subtask are given based on annotations on test methods and classes. For example, if a test class has an annotation `@Points("007")`, the student is awarded points for the exercise "007", given that all the tests in the class pass, and no other class or test with the same annotation fails.

The exercise is published by adding it to the course git repository and refreshing the course in TMC. This causes a HTTP push event, which can be identified by the NetBeans plugin. Alternatively, the exercise (or an update to the exercise) is available the next time the student opens up the IDE.

Instructors may choose to have a separate branch for exercises that need to be tested internally before publishing. One can e.g. create a separate course for more eager students, who are willing to work as exercise testers. Once a set of exercises is tested well enough, and deemed ready for publishing, the branch is merged to the branch that is visible to the whole course. This is especially useful when performing team teaching and collaborative crafting of material.

3.3 Deterministic Profiling

Algorithm-related tests are usually assessed simply by investigating the algorithm run-time with different sized inputs. This is problematic as operating system-related tasks and e.g. JVM garbage collection tasks launch arbitrarily, causing additional load on the assessment system. This may lead to false negatives, even if the assessed algorithm is implemented perfectly. One common approach is to average the running time over a specific number of iterations. However, this simply tries to avoid the actual problem: assessing a program based on execution time.

TMC has a bytecode counting component that is inspired by ByCounter [12]. Counting bytecode instructions makes it possible to conduct repeatable benchmarks of students' algorithms. One can demand e.g. that an algorithm must run in a linear time or faster based on the size of an input.

Setting up a deterministic test needs an input for the algorithm and a method to invoke. In the following example, students need to create an algorithm for calculating the Fibonacci numbers that is linear to its input size or faster. The method must be created to a method called `fibonacci` in a class called `Fibonacci`. The method takes an integer as a parameter, and returns an Integer as output. The `IntegerImpl` is a wrapper that expects an integer as a return value from the method.

```
@Test
public void linearFibonacci() {
    IntegerImpl impl = new IntegerImpl();
    impl.setClassName("Fibonacci");
    impl.setMethodName("fibonacci");

    List<Integer> input = Arrays.asList(2, 10, 100);
    Output<Integer> output = impl.runMethod(input);
    ComplexityAnalysis.assertLinear(output);
}
```

The complexity analysis component also provides a graph view which allows easy visualization of the algorithm running times.

4 Scaffolding Students with TMC

The driving learning method in XA is individual effort through practical work by students. This means that the exercises are the most important part of a course, and designing them is the most important task for the teacher in charge. Every instructional goal in a course is learned via working through a set of exercises designed to help building understanding on the topic at hand. Being able to solve an exercise is not enough: one must focus on both the process and quality while crafting the solution. Course instructors monitor and help students as they work on the exercises, which helps students in achieving their goals as well as provides feedback for the instructors on upcoming exercises that should be created [17].

Exercises that the students start with are usually composed of small incremental tasks, which combine into bigger programs. Incremental tasks are used to imitate a typical problem-solving process: as the students work through the tasks, they explicitly practice building software from smaller components.

The written-out thought process that was used to form the exercises constantly influence the students' programming. This provides scaffolding for learning of good programming practices, as students' work is constantly guided by the pre-performed subtask division. Exercises are intentionally written out to be as informative as possible, and often contain sample input/output descriptions or code snippets with expected outputs, which provide further support for verifying the correctness of a crafted program.

As an example of a scaffolding assignment, let us examine a sequence of exercises that demonstrates the use of methods. Just before the assignment, the course material presents the use of void methods and how a method can call another user-defined method. Before this set students solve one simple warm-up assignment involving only parameterless methods. The assignment belongs to the 2nd week of our 14-week CS1 course, and the students have practiced with loops and variables starting from week 1.

Assignment: Printing a Christmas tree

Task 1: Printing stars Modify the method `printStars` so that it prints the given amount of stars and a line break. Use the following body:

```
private static void printStars(int amount) {
   // you can print one star with System.out.print("*");
   // call it 'amount' times
}

public static void main(String[] args) {
    printStars(2);
    printStars(9);
}
```

The program should output:

```
**
*********
```

Task 2: Printing a rectangle Create a `printRectangle(int width, int height)`-method that prints a rectangle using the `printStars` method. Calling `printRectangle(17,2)` should produce the following output:

```
*****************
*****************
```

Task 3: Printing a left-aligned triangle Create a method `printLeftAlignedTriangle(int size)` that prints a triangle using the method `printStars`. Calling the new method with `3` as a parameter should produce the following output:

```
*
**
***
```

Task 4: Printing stars and whitespaces *omitted*

Task 5: Printing a right-aligned triangle *omitted*

Task 6: The tree Create a method called `xmasTree(int height)` that prints a Christmas tree using at least some of the previously defined methods. A Christmas tree consists of a triangle of given height and a stand. The stand is a single star located at the middle of the triangle bottom. The method call `xmasTree(3)`, for example, has the following output:

```
    *
   ***
  *****
    *
```

While performing the steps in the above exercise, students practice creating and using methods with parameters, work constantly using a divide-and-conquer approach, and see how a simple algorithmic challenge, e.g. printing a tree, is solved.

Scaffolding within the exercises can also be used to direct the students away from bad habits such as the use of unnecessary instance variables or unclear method names. As the students work on the exercises, their workflow resembles the workflow of TDD [2] that the instructor that created the exercise previously had followed. However, in most of the cases, the tests are now pre-defined. A clear metalevel motivation for the incremental style is to guide students to follow a working process similar to that of good professional programmers: proceed in small steps and validate your code after each step [2].

The scaffolding is implemented in the tests of the exercises. The tests are written so that they help students to focus on progressing in small steps even within a single exercise: it is important to concentrate on making tests pass one by one in a meaningful order. A typical sequence might be:

1. implement class MainProgram
2. define a static method printStars(int amount)
3. ensure that the method prints the correct amount of stars
4. ensure that a newline is printed after the stars
5. define a static method printRectangle(int width, int height)
6. ensure that the method prints the correct amount of stars when called as print-Rectangle(1, 1)
7. ensure that printRectangle(1,1) calls printStars(1)
8. . . .

4.1 Open Exercises

As any instruction should aim towards the student being able to do the problem solving themselves, it is important that scaffolding is eventually faded [4]. In our programming courses, fading is realized by using open exercises that do not enforce any specific program structure or approach.

Open assignments in early programming classes are intentionally complex e-nough so that programming a solution to a single file (e.g. class) causes chaotic design, but simple enough so that using an “implement a single requirement, refactor if needed” approach will eventually create a nice object design. The exercises often utilize a well-known domain (e.g. airport, airplanes or student, courses, registrations), which makes it easier to design required domain objects.

The following exercise is an example of an open exercise from week 6. In addition to the problem description, the students receive a sample input/output description, which has been left out due to space considerations.

Assignment: Bird Observations

Design and implement an observation database for a bird observer. The database contains birds, each of which have name and latin name. The database also tracks how many times a bird has been observed. The program should have a text UI and should respond to the following commands

```
add - adds a bird
observation - adds an observation
statistics - prints all birds and observations
show - prints one bird
quit - terminates the program
```

The database should store the observations into a text file for future use.

The open exercises only define how the application is supposed to work for a given user input. In early programming courses the input may be given e.g. via command line, or as system input, while in web-development courses the tests for open exercises typically monitor e.g. a given database while inputting data to input fields with specified names or ids, or require that a given REST API exists and works as desired.

5 Conclusions and Further Work

The automated assessment system, TMC, has been successfully used in our CS1, CS2 and Web-development courses, as well as in various other courses, including MOOCs [15] in programming. We have also utilized TMC and NetBeans for younger students in a "game programming for youth" outreach course that has been created as an attempt to raise awareness towards programming. The youngest students so far have been 11, and given the challenges of learning Java at a young age, TMC itself has worked well.

As TMC can provide some of the scaffolding for the students to learn programming, it has allowed better allocation and use of resources in our courses. Instructors now spend more time on more demanding scaffolding and have time to reflect in the labs when compared to the past, where the exercises were checked manually. This has contributed to improvements in the learning results in our CS1 courses, and made the task of working as a TA a valuable learning experience [17].

The blended learning environment that TMC and XA creates resembles the genuine working environment of a software developer. Our students are immersed in this environment from day one of their studies, working with industry strength tools and pushed towards programming best practices. As TMC is not a course or topic specific tool, it can be used in a similar fashion throughout our curriculum. The following spontaneous testimonial is from a web backend development course held during fall 2012.

> TMC was great! It was great to be able to work on so many web applications. Receiving many of the configuration files with tests that said what to do aided a lot during learning. In addition, it was good that I didn't have to waste the time of TAs for showing exercises that I was confident about. TMC was very easy to install, and incredibly easy to use.

Currently TMC supports Java (and other JVM-based languages), and we are in the process of developing a more modular, language-independent, assessment backend, as well as integration to other IDEs. We are considering integrating static analysis tools such as PMD and Checkstyle with TMC, which would enable us to better address code conventions (eg. indentation, variable naming, method length and complexity) that are currently evaluated in the labs in an ad-hoc fashion. In addition, a component that flags submissions that are potentially copied from other students or model answers is under development.

We continuously wish to improve our CS education. Analyzing snapshot data from the programming process of our students, gathered by TMC, will help to improve both the exercises and the accompanied tests in our courses. This will benefit our students and teams of teachers as the learning of the students is better understood and can therefore be more effectively scaffolded.

6 Acknowledgements

We thank all the students who have worked on building TMC so far: S. Hiltunen, J. Isotalo, K. Kaltiainen, V. Knuuttila, T. Koivisto, T. Kovanen, A. Majander, P. Marjanen, J. Mynttinen, K. Nordman, M. Rannanjärvi, K. Rantanen, O. Rissanen, T. Simsiö, J. Turpeinen and K. Viiri.

References

[1] K. Ala-Mutka. A survey of automated assessment approaches for programming assignments. *Computer Science Education*, 15(2):83–102, 2005.

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

[3] J. Brown, A. Collins, and P. Duguid. Situated cognition culture of learning. *Educational Researcher*, 18(1):32, 1989.

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

[5] A. Collins and J. G. Greeno. Situative view of learning. In V. G. Aukrust, editor, *Learning and Cognition*, pages 64–68. Elsevier Science, 2010.

[6] C. Douce, D. Livingstone, and J. Orwell. Automatic test-based assessment of programming: A review. *Journal of Educational Resources in Compututing*, 5(3), Sept. 2005.

[7] S. Edwards. Using test-driven development in the classroom: Providing students with automatic, concrete feedback on performance. In *Proceedings of the EISTA'03*, volume 3. Citeseer, 2003.

[8] M. E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3):182 –211, 1976.

[9] J. G. Greeno. Response: On claims that answer the wrong questions. *Educ. Researcher*, 26(1):5–17, 1997.

[10] D. Hovemeyer and W. Pugh. Finding bugs is easy. *SIGPLAN Not.*, 39(12):92–106, Dec. 2004.

[11] P. Ihantola, T. Ahoniemi, V. Karavirta, and O. Seppälä. Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling*. ACM, 2010.

[12] M. Kuperberg, M. Krogmann, and R. Reussner. ByCounter: portable runtime counting of bytecode instructions and method invocations. In *Proceedings of the ETAPS'08*, 2008.

[13] 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.

[14] J. Spacco, D. Hovemeyer, W. Pugh, F. Emad, J. K. Hollingsworth, and N. Padua-Perez. Experiences with Marmoset: designing and using an advanced submission and testing system for programming courses. In *Proceedings of the ITICSE '06*. ACM, 2006.

[15] A. Vihavainen, J. Kurhila, and M. Luukkainen. Multi-faceted support for MOOC in programming. In *Proceedings of the SIGITE '12*. ACM, 2012.

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

[17] A. Vihavainen, T. Vikberg, M. Luukkainen, and J. Kurhila. Massive increase in eager TAs: Experiences from extreme apprenticeship-based CS1. To appear in *Proceedings of the ITiCSE'13*, July 2013.

[18] D. Wood, J. S. Bruner, and G. Ross. The role of tutoring in problem solving. *The Journal of Child Psychology and Psychiatry and Allied Disciplines*, 17(2):89–100, 1976.