Teaching Programming to Large Student Groups through Test Driven Development
Comparing Established Methods with Teaching based on Test Driven Development

Morten Goodwin[1] and Tom Drange[2]
[1]*Institute of Technology and Sciences, University of Agder, Kristiansand, Norway*
[2]*Noroff University College, Kongsvinger, Norway*

Keywords: Programming, Test-Driven Development, Automatic Assessment, Education.

Abstract: This paper presents an approach for teaching programming in large university classes based on test driven development (TDD) methods. The approach aims at giving the students an industry-like environment already in their education and introduces full automation and feedback programming classes through unit testing. The focus for this paper is to compare the novel approach with existing teaching methods. It does so by comparing introduction to programming classes in two institutions. One university ran a TDD teaching process with fully automated assessments and feedback, while the other ran a more traditional on-line environment with manual assessments and feedback. The TDD approach has clear advantages when it comes to learning programming as it is done in the industry, including being familiar with tools and approaches used. However, it lacks ways of dealing with cheating and stimulating creativity in student submissions.

1 INTRODUCTION

Teaching programming to large students groups is challenging (Schulte and Bennedsen, 2006; Milne and Rowe, 2002; Barr and Guzdial, 2015). One of the reasons for this is the group diversity which typically leads to teaching methods fitting well for a subgroup of the students, but not all. Additionally, the number of students makes it challenging for teachers to spend sufficient time with each student in order to facilitate good learning.

When students do a programming assignment in most universities, they work in an offline process. They will upload their submissions to a Learning Management System (LMS) and await approval from the teacher. This asynchronous environment has several challenges, namely that it: (1) Forces a long waiting period between student's work and feedback. (2) Makes the teacher do robot-like tasks such as correcting many assignments instead of spending quality time with the students. (3) The work has very little resemblance to the industry practices for which the universities are supposed to prepare students.

Test-driven development is the de-facto standard for software development and testing in the industry today. This centers on writing small pieces of code to test the software, and in turn running the tests automatically to verify the correctness of ongoing and delivered products.

T-FLIP[1] is a project that aims at, among others, making the industry like environment available when teaching programming. This paper compares two vastly different teaching approaches: (1) Teaching programming through TDD in an industry like model, and (2) Teaching programming in a more traditional on-line environment with tutors and manually assessed course work. This is done by comparing two practically very similar courses carried out with the two models.

The outline of the article is a follows. Section 2 introduces the T-FLIP project and motivates the use of an industry like environment through automated assessment. Section 3 continues with a more traditional approach for learning programming in an on-line environment. Section 4 compares the two approaches and pinpoints advantages and disadvantages. Finally, section 5 discusses the findings and section 6 concludes the work.[2]

[1]http://tflip.uia.no

[2]A preliminary version with some of the results is in the process of being published as a chapter in a book titled 'Digital media, tools, and approaches in teaching and their added value'. This includes presentation of the T-FLIP project. (Goodwin et al., 2015)

2 LEARNING PROGRAMMING WITH TEST DRIVEN DEVELOPMENT (TDD)

This section introduces a teaching method using TDD as a corner stone for the student activities. This method is applied in programming courses at University of Agder, Norway.

There is an extensive amount of literature covering how to best teach programming (Mayer, 2013). There is, however, only marginal works on methods that enable students to work in industry-like environments with automated feedback in order to improve their skills (Boydens et al., 2015). The T-FLIP project work on an environment for automated industry like feedback in programming (Goodwin et al., 2015). It addresses how to best facilitate effective learning for large student groups in programming courses with automation in industry-like environments. They do this through an environment where the students can upload their assignments and receive direct, automated feedback on whether they have passed the assignments given to them. The environment is designed to be similar to the industry students will meet after graduating.

2.1 Motivation

The motivation for learning through TDD in a study is to look into key factors that facilitate learning mechanisms in programming courses, and how it is possible to provide better and more realistic learning environments for the students. This is in line with the well established "Learning by doing" principal and experience-based learning (Gentry, 1990; Ahmed, 2015) aimed at establishing patterns of action that come about by repeated exercise in a realistic software development environment.

More specifically, T-FLIP investigates how to:

1. Best enable the students to actively work in an industry environment.
2. Allow for automated constructive feedback.
3. Allow for constructive feedback through peer review analysis.

2.1.1 Business Environment

The favored method today is test-driven development with smaller teams using agile methods and automated testing (Boydens et al., 2015; Canfora et al., 2006).

For the students, the transfer of learning is naturally significantly higher when working in a context similar to the one they will face later (Perkins and Salomon, 1992). Hence, not focusing on industry tools in university education could potentially become a problem for future students, who, upon graduating, will lack practical experience. Introducing students to assignments closely linked to the actual work environment they will encounter, might improve the learning (Perkins and Salomon, 1992) . T-FLIP aspires to giving students practical use of methodology and tools that will prepare them for their future programming work situations.

2.1.2 Immediate Feedback

Successfully teaching students programming depends on producing feedback on the students performance. Students find it difficult to understand their performance if they do not have anything to measure against (Lachman, 2015).

Therefore, it is not surprising that best practice in teaching programming courses dictates constant interaction between the teacher and student (Eckerdal et al., 2005). This immediate feedback received in a close teacher/student environment is an important encouragement for the student to increase and maintain motivation, and is an essential part of their work for better results.

Following this line of reasoning, an essential environment for learning programming should yield immediate feedback on the students work as a step towards a more rapid understanding of what they are doing. This is difficult to achieve since most university programming courses consists of large groups and limited resources for each student.

This immediate student feedback mirror the automated feedback received in TDD.

2.1.3 T-FLIP

A prototype is created to test the feasibility of the industry-like programming environment for the students. This prototype has been designed using well known industry software tools intended to be an almost complete industry-like environment both for teachers and students.

The prototype is used to some extent in six programming courses at the University of Agder , Norway. The usage varies from only enabling version control, to a fully functional environment with automated testing. For reasons of comparison, this paper mainly addresses the basic programming course.

In the developed prototype, we have chosen to implement the two most important aspects to get an industry look-and-feel, namely the use of industry tools and immediate feedback.

2.1.4 Prototype

The prototype is developed using well-known software development tools provided by the vendor Atlassian, which produces software for software developers and project managers. Note that the objective is for the students to learn general skills that they can apply to other alternative tools, including tools from other companies. Atlassian was chosen based on the following main criteria: maturity, usage by the industry, freely available for university use, and the availability of client tools. It is important that the tool reuses existing technologies for three main reasons:

1. Facilitating familiarity with the tools for the students.
2. Utilize existing functionality and use already existing software development environments.
3. Reduce maintenance.

The prototype consists of 3 main components:

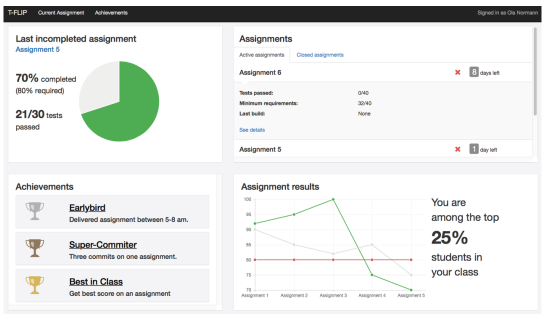
1. Version Control Repository Stash/git.
2. Build and test environment Bamboo.
3. Build Agents

The tools server is the interface through which the students and teacher interact with a repository (git environment) and an interactive web site. In addition, the tools server provides an interactive website that the students can use to see their committed code and start the build process.

Upon testing the assignments, the students push to the repository, and activate the build process in the tools server (see Build and Test Environment). Building means compiling the code and running the tests. Upon building, a course specific build plan (see Build Plan) is activated and spawned on a separate build server. The build agents compile, test and give feedback back to the student and teacher. The student interaction resembles businesslike software development activities (see Student Activities). Further, later versions have the possibility to include static analysis (see Additional Feedback) and peer review (see Peer Review).

It is essential programming practice to be familiarity with version control. In our system, each student is given his/her own git repository in an Atlassian Stash environment. In the prototype, the only way to deliver student assignments is to commit and push their works on their repository, making this a way to get practical experience with git in a learning-by-doing fashion.

Figure 1 presents a prototype of the user interface presented to the students.


T-FLIP Current Assignment Achievements Signed in as Ola Normann

Last incompleted assignment
Assignment 5
70% completed
(80% required)
21/30 tests passed

Assignments
Active assignments Closed assignments

Assignment 6	✖	8 days left
Tests passed:	0/40	
Minimum requirements:	32/40	
Last build:	None	
See details		
Assignment 5	✖	1 day left

Achievements
Earlybird
Delivered assignment between 5-8 am.
Super-Commiter
Three commits on one assignment.
Best in Class
Get best score on an assignment

Assignment results
You are among the top 25% students in your class

Figure 1: Prototype of GUI for T-FLIP.

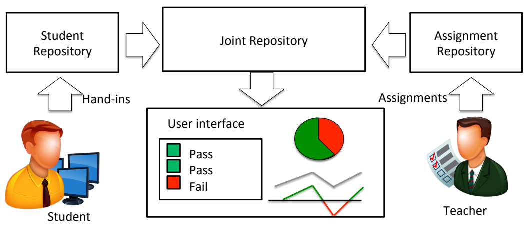


Figure 2: Build process for T-FLIP showing how teachers push assignments and test on one repository, and students results on another repository. In turn the repositories are merged, built and results made available to teacher and student.

2.1.5 Build and Test Environment

To evaluate programming assignments automatically, there is a need to build the assignments and run the corresponding tests. This functionality is achieved for each course by having a number of repositories (see Figure 2):

1. Student repository: git repositories where the student commits assignment submissions.
2. Assignment repository: git repository where the course responsible commits unit tests for each assignment.
3. Joint repository: git repository that combines the two above repositories to apply the unit tests.

Prior to any student work, the teacher creates tests for each assignment available in the teacher repository. When the students carry out the assignments, they commit and push them in to their student repository. During the build process (see Figure 2), the student and teacher repositories are merged in into the combined repository,. This enables testing of the student assignments without the student having to access the unit tests actively. The entire process is run in Atlassian Bamboo allowing a dynamic, relatively easy to use, build environment for students and teachers. An important concept is that a successful student assignment will build successfully and pass all unit tests given by the teacher, while an unsuccessful assignment will have one or more failing tests. Hence, the students can know whether they have successfully carried out an assignment by simply running the unit tests and receiving immediate feedback. The proposed methods allow this to be possible with limited additional work from both the teacher and student.

2.2 Build Process

The build plan is a formal description of how to assemble and test the software specifically for the course. The steps of the build process are as follows, in line with the illustration in Figure 2:

1. Initially, the repositories are merged and build as described in Version Control
2. Subsequently, the student assignments are cleaned up to make sure they fit the tests. Cleaned in this context means adding proper packaging information, moving to correct folder, and so on. Unit tests have strict requirements on class and method names. It further replaces some standard classes with mock-up variants, which are easier to test. The latter is a commonly used technique in unit testing
3. Compiling the code means making the code machine runnable, which is a requirement to both run the application and run the tests.
4. Running the unit tests is the process of applying the teacher tests on the student assignments.
5. In this context, static analysis refer to tests that are not related to the assignment, but more related to the code in general. For example is the code readable, and are all variables used?
6. Step 4. and 5. output XML-based results. All XML-results are translated to an agreed upon JSON-structure that is easily parsable by the front end. Any additional modules, such as peer review and plagiarism detection should follow the same structure.
7. The front end parses the JSON data and presents it as an interactive web site (see Figure 1) including feedback on individual assignments.

   The front end The last step parses the XML into readable web pages for the students.

2.2.1 Student Activities

The student is not required to know all details of the build plan. From the student point of view, T-FLIP is an interactive system that for all intents and purposes is like the build process in the industry.

The typical steps the students follow are:

1. Do the required assignment.
2. Commit and push the files to the student repository.
3. Build the assignment (including tests from the teacher) on Bamboo.[3]
4. Read the results.
5. If all tests pass, you are finished with the assignments. If at least one test fails, you need to go back to step one.

There are several benefits when organizing it like this. Firstly the students get instant feedback on how they are doing. Secondly, students are not able to read the teacher repository and can therefore not cheat by knowing the details of the unit tests.

3 LEARNING PROGRAMMING IN AN ON-LINE ENVIRONMENT

For comparison of the TDD teaching approach, this section introduces a well established approach for learning programming in an on-line environment. This is carried out at Noroff University College, Norway.

3.1 Overview

Introduction to Programming runs over seven weeks. It is delivered symmetrically to campus and online based students through the use of learning elements presented through a Learning Management System. Video streaming equipment and real-time chats are used to deliver lectures to these students simultaneously. There are in total 3x45 minutes per week of teaching and 3x45 minutes per week of lab work during the course.

3.2 Learning Elements

The learning elements include Lecture Presentations - slides that the lecturer uses during the lecture, Recordings - all lectures are recorded for later review, Activities - activities that the students are encouraged to work through as that provides the practical training in this course and Resources - tools needed for the course are published, such as compilers, test-files, links to resources on the Internet and so on.

[3]The reason we do not use git-hooks is to avoid excessive activity on the server if the students do many consecutive pushes.

3.2.1 Activities

The activities varies in difficulty as the course progresses, and can involve anything from answering theoretical questions about programming and the thought processes used, to actual programming tasks to be executed. Responses to the tasks in the activities is then uploaded by the students through LMS and feedback is also given through LMS.

3.3 Assessments

The assessments are divided into four parts, three course work assignments and a reflective journal in the format of a blog. The latter is not relevant for this paper and will not be discussed here.

4 BASIC PROGRAMMING — A COMPARISON

This section compares the basic programming course applied with T-FLIP with the more traditional symmetrical on-line course. The aim is to better understand the differences between applying a course using the T-FLIP method and a more traditional course that is run on-line and on-campus.

The courses included are two very similar basic programming courses at two close-by institutions in Norway. University of Agder has carried out their introduction to programming course to fit TDD in the industry. For reasons of brevity, we refer to this course as (T-FLIP). Noroff University College uses a more standard approach teaching programming through programming assignments and blogs which are manually evaluated by the teachers involved. For reasons of brevity, we refer to this course as (Noroff)

Table 1 presents an overview of the courses involved in the study.

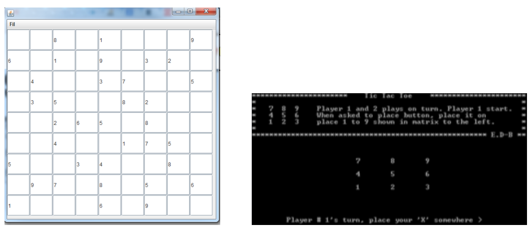
4.1 Introduction to Programming at (T-FLIP)

This section presents an introduction to programming course using the TDD method of teaching.

This course teaches Java and consists of, among other activities, 16 weekly assignments ranging in difficulty from “hello world-applications to desktop games such as Snake and Pacman. At least 200 participating students actively use the prototype weekly.

4.1.1 Example of Assignment and Feedback

This section presents an example of an assignment, and a practical example of feedback in (T-FLIP).



```
*********************    Tic Tac Toe    *********************
*                                                           *
*   7  8  9    Player 1 and 2 plays on turn. Player 1 start.*
*   4  5  6    When asked to place button, place it on      *
*   1  2  3    place 1 to 9 shown in matrix to the left.    *
*                                                           *
*************************************************** E.D-B **

                      7        8        9

                      4        5        6

                      1        2        3

          Player # 1's turn, place your 'X' somewhere >
```

a) (T-FLIP) b) (Noroff)

Figure 3: Example of assignments.

Figure 3(a) presents a student assignment where the student is supposed to create the small game of snake. This is the 13th assignment in the course, and it is split into to several smaller assignments, which correspond to unit tests. The first four assignments are, given to the students, are: (abbreviated for reasons of readability):

- (Part 1) Make a graphical user interface with 9x9 text fields.
- (Part 2) Make a menu.
- (Part 3) Implement so that users can generate a new game.
- (Part 4) Implement so that users can restart games.
- More advanced parts are are removed for reasons of brevity.

Following the description of the assignment, the students need to develop Java classes with classes methods and methods as they see fit. Further, the students need to write part of the application responds to keyboard actions.

The test, written by the teacher, checks if the correct components are used in the including existence of 81 text fields (Part 1), existence of a menu (Part 2) , checks that a new correct game is generated when pressing for a new game (part 3), and checks that the same game is restored when pressing for restart. (Part 4).

As indicated in Figure 2, after the students have implemented the assignment they commit and push the application to their repositories and build the application. The student gets an interface presented after the build is finished.

4.1.2 Interactive System

(T-FLIP) was actively used by student the course. If a students pushes code and builds the application so that

Table 1: Courses part of the comparison.

Short reference	Course	ECTS	Institution	Main teaching method
(T-FLIP)	Introduction to Object Oriented Programming	10	University of Agder	TDD on-campus with T-FLIP
(Noroff)	Introduction to Programming	10	Noroff University College	Manual assessment on-campus on-line

the tests pass, their implementation is correct and they can continue with the next assignment. In contrast, if one or more tests fail, it indicates that the student has not carried out the assignment correctly and need to redo their assignment. This is repeated until all they pass all their tests. In this way, the the students are guided towards the correct solution without constant need of teacher interaction.

As an added bonus, the students could work at any hour without having to rely on the teacher. Further, the teacher could make additional assignments without needing to increase the use of resources in correcting.

4.1.3 Formal Process

Programming is a very formal process. As an example, understanding the potential impact of a missing parenthesis in an application is not easy to grasp for several of the students.

Since (T-FLIP) is a very formal feedback process, students never passed if not the entire code was correct. Following the same example as above, missing a (crucial) parenthesis might be acceptable by a teacher as a minor issue, but it not accepted by (T-FLIP). This is in line with a build environment in the business world; It could very well be that minor issues with some parts of the code breaks the entire program, and pushing broken code is not correct.

We experienced that the students quickly understood this since (T-FLIP) does not separate between minor issue or major issue breaking the code.

4.1.4 Use of Git Repositories

For the initial assignments, students generally struggled with the git-process. The students did not get any training in git, and in turn felt git was difficult. Very few had experience with git prior to the course.

Many students committed wrong files, forgot to push some files, and made multiple repositories that conflicted with each other. These problems almost magically vanished as the students became more familiar with the tools, indicated that the learning by doing had a great impact on git.

4.1.5 Possibilities of Cheating

In (T-FLIP) the teacher has little control over what the students actually commits. It is easy for a student to copy code from another and push it as their own without the teacher ever knowing about it. In all fairness, in large classes with small assignments this is always possible, but it is probably easy to hide behind automated approaches since no human is involved in the process. A possible mitigation is to implement cheat detection through e.g. similarity measurements of assignments (Wilcox, 2015; Cheang et al., 2003) that can indicate if two assignments are too similar.

4.1.6 Tests which are not Passed in the Automatic Testing

Some students choose to go beyond the actual assignment given by implementing new, often times fancy, features of their own liking. Ideally, this should be encouraged since students can enjoy programming when they implement something they have decided themselves. However, the automated tests, which are very strict, will most often fail as the student is not properly in line with the assignment. It becomes a trade-of between formality and creativity which causes practical difficulties it is not always easy to write unit tests for creative solutions.

4.1.7 Learning Outcome

The students work in a TDD environment intended to get a "feel" the industry way of working in a development project. It answers the need for system development students to learn about testing (Sheth et al., 2012). Because the students work in a TDD environment, they become used to TDD including functional experience with practical tools such as git and build tools. Hopefully, the TDD experience gives them an advantage when starting in the industry.

4.2 Introduction to Programming at (Noroff)

This section presents an introduction to programming course using the established on-line method of teaching.

4.2.1 Example of Assignment and Feedback

This section presents an example of an assignment, and a practical example of feedback in (Noroff).

Figure 3(b) presents the second of three course work assignments, presented to the students in the fourth week out of seven. It is split into two main tasks, creating a birthday book and a tic-tac-toe game.

The tic-tac-toe game should use a 2D array to record the moves and print appropriate outputs to the screen to enable two people to play the game. The students does not have to create computerized opponents.

Both parts of the assignment should be included in a word processed report that presents the solutions to the tasks, by including code and screen shots of the programs in action.The students must also include a brief commentary on potential challenges and how these were worked through and hopefully overcome. Completed assignments are then uploaded to the LMS.

It is noteworthy that the tic-tac-toe game in (Noroff) — Figure 3(b), has many similarities with the Sudoku game in (T-FLIP) — Figure 3(a).

4.2.2 Possibilities of Cheating

In (Noroff) the teacher has full control over each uploaded assignment, and will be able to run checks for plagiarism and similarities. In large classes with many small assignments, it is as earlier mentioned always a risk that students finds a way to cheat, but at least there is a human involved to spot potential irregularities.

4.2.3 Students that go Beyond Actual Assignments

Since there are teachers involved in the assessment, students who choose to go beyond the actual assignments may be rewarded for this, as the teacher will see the extra effort and creativity. Dunlap (2005) argued that students that are able to learn and adapt because of their reflection and their willingness to ”...go beyond what they know ...” (Dunlap, 2005). This is difficult to achieve in in an automated system.

4.2.4 Learning Outcome

The students will not get the same ”feel” for the industry way of working in a development project as the students in (T-FLIP) will. Toeffler (1970) said very early that we should teach the individuals to teach themselves, and considering that the important thing is not what we as teachers do, but what the students does, leaves us to believe that the many weekly activities, the course work and the blog assessments will allow students to obtain the learning outcomes (Biggs, 1999; Toffler, 1990). However, we must not forget that students will not learn if they are not motivated, so regardless of the methods we use, motivation should be a major part of our efforts in teaching programming (Jenkins, 2001).

5 DISCUSSION

The comparison of the two approaches to teaching programming shows both advantages and disadvantages with both systems. The actual way the courses are delivered are very similar with both lecturing and lab work run by a teacher. The activities and assignments and the way they correspond with the topic being taught in addition to the increased level of complexity, are also similar i both approaches. Two major benefits of (T-FLIP) is the industry-like feeling the students get by working with automated feedback, and the extra time the staff gets by not having to manually follow up every question surrounding the student assignments. Two major benefits of (Noroff) are the personal “hunch” the teachers get about cheating when they manually review the assignments, and the possibility for students to go beyond the actual tasks and get credit for that.

An implementation of cheat detection tools could be a way of mitigating the challenge of detecting cheating students, but such tools can be implemented regardless of teaching method used. The (T-FLIP) will not be able to accommodate eager students going beyond the actual assignments in the same way as teachers manually reviewing tasks. However the fact that the automated system will force the students to correct minor errors in the code, such as missing semi colon or misspelled arguments, gives the students an industry-like experience that the teacher is not able to accomplish with manual reviewing.

Table 2: Comparison between (T-FLIP) and (Noroff).

Property	T-FLIP	Noroff
Possible to get immediate feedback	High	Low
Possible to get feedback outside of regular hours	High	Low
Work in and learn an industry setting	High	Low
Learn formal setting of software project building	High	Low
Possibility to try again after failing a task	High	Medium
Avoid Plagiarism	Low	High
Encourage going beyond actual assignment	Low	High

Table 2 presents an overview of the comparisons between (T-FLIP) and (Noroff) where High is generally a positive aspect, and low is a negative aspect.

6 CONCLUSION

The system is a success; the students are actively using the prototype and are already in their education working and becoming familiar with tools almost exactly as they are used in the system development industry. Further, the students get automated feedback, which motivates them in lab classes. Due to the automation, they get prompt response on their work without having to rely upon the teachers available time, which is in line with best teaching practices for programming courses. There are however some advantages to the manual reviewing of assignments, which indicates that a combination of elements from (T-FLIP) and (Noroff) would probably provide the best learning environment for students learning programming.

ACKNOWLEDGEMENTS

The T-FLIP project is co-funded by Norgesuniversitet P102/2015 http://tflip.uia.no.

REFERENCES

Ahmed, M. (2015). Effectiveness of tdd on unit testing practice.

Barr, V. and Guzdial, M. (2015). Advice on teaching cs, and the learnability of programming languages. *Communications of the ACM*, 58(3):8–9.

Biggs, J. (1999). What the student does: teaching for enhanced learning. *Higher education research & development*, 18(1):57–75.

Boydens, J., Cordemans, P., and Hallez, H. (2015). On using test-driven development to tutor novice engineering students using self-assessment. In *Proceedings of the 43rd SEFI Annual Conference*, pages 182–182. SEFI-Société Européenne pour la Formation des Ingénieurs.

Canfora, G., Cimitile, A., Garcia, F., Piattini, M., and Visaggio, C. A. (2006). Evaluating advantages of test driven development: a controlled experiment with professionals. In *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pages 364–371. ACM.

Cheang, B., Kurnia, A., Lim, A., and Oon, W.-C. (2003). On automated grading of programming assignments in an academic institution. *Computers & Education*, 41(2):121–131.

Dunlap, J. C. (2005). Changes in students' use of lifelong learning skills during a problem-based learning project. *Performance Improvement Quarterly*, 18(1):5–33.

Eckerdal, A., Thuné, M., and Berglund, A. (2005). What does it take to learn'programming thinking'? In *Proceedings of the first international workshop on Computing education research*, pages 135–142. ACM.

Gentry, J. (1990). What is experiential learning? guide to business gaming and experiential learning. *J. W. Gentry, Association for Business Simulation and Experiential Learning (ABSEL)*.

Goodwin, M., Auby, C., Andersen, R., and Barstad, V. (2015). Educating programming students for the industry. *Digital Media in Teaching and its Added Value*, page 100.

Jenkins, T. (2001). Teaching programming–a journey from teacher to motivator. In *2nd Annual LTSN-ICS Conference*.

Lachman, N. (2015). Giving feedback to students. In *Teaching Anatomy*, pages 143–153. Springer.

Mayer, R. E. (2013). *Teaching and learning computer programming: Multiple research perspectives*. Routledge.

Milne, I. and Rowe, G. (2002). Difficulties in learning and teaching programmingviews of students and tutors. *Education and Information technologies*, 7(1):55–66.

Perkins, D. N. and Salomon, G. (1992). Transfer of learning. *International encyclopedia of education*, 2.

Schulte, C. and Bennedsen, J. (2006). What do teachers teach in introductory programming? In *Proceedings of the second international workshop on Computing education research*, pages 17–28. ACM.

Sheth, S. K., Bell, J. S., and Kaiser, G. E. (2012). Increasing student engagement in software engineering with gamification.

Toffler, A. (1990). *Future shock*. Bantam.

Wilcox, C. (2015). The role of automation in undergraduate computer science education. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, pages 90–95. ACM.