Teaching Software Testing in an Algorithms and Data Structures Course

Andrea Arcuri
Kristiania University College
Department of Technology
Oslo, Norway

***Abstract*---In this paper, we report and share our experience on introducing the teaching of *Software Testing* in an *Algorithms and Data Structures* course. The goal was twofold: (1) help in the teaching of the complex subject of algorithms and data structures and (2) start to make the student familiar with the topic of software testing. We report on the lessons learned in such endeavor and on the results of a student survey, after the course was completed. How the teaching of software testing was introduced was positively evaluated by the students. All our teaching material is freely available online on GitHub.**

***Index Terms*---Software testing, test education, unit testing, test course**

I. Introduction

Algorithms and Data Structures (''*AlgDat*'', in short) is a common course in Computer Science (CS) and Software Engineering (SE) programs, taught in many universities around the globe. Typically, in such a course, students learn the low details of the fundamental data structures in programming, such as *Lists*, *Maps*, *Sets* and *Graphs*. Algorithms like *Sorting* (e.g., *Merge* and *Quick Sort*) are common as well to be taught in such a course. Not only the students are taught how such algorithms are implemented, but also how to analyze their performance and asymptotic runtime based on the input size. For example, students learns how to use the *Big O* notation for upper-bounds, and Ω for lower-bounds.

AlgDat is typically considered a challenging topic, especially when taking into account that its proper understanding does require a strong mathematical background (which unfortunately many students lack). A possible issue is that students might not fully appreciate the importance of a topic when they do not use its results in practice. While a young software engineer in training might be inspired by learning how to develop a mobile or web application, they might be less thrilled when learning some data structure and its mathematical properties without any actually need yet to use it. SE is a practical field, like any engineering fields. And most students that enroll even in CS degrees will work as engineers after graduation, and not as computer scientists [1]. Learning the theoretical foundations is important, but it is also important to do not forget the practical and engineering aspects of *AlgDat*.

To cope with such issues, a possibility is to introduce the students to the topic of *Software Testing* [2], in particular *Unit Testing*. In the context of *AlgDat*, having the course material including unit tests does bring the following benefits:

- A test case reppresents a *usage scenario* of an algorithm / data structure, which might help understanding how to use them.
- Test cases can be run in *debug* mode, where students can see step-by-step how the algorithms operate and how the memory is manipulated (e.g., the object *Heap* and the variables in the frames of the *Function-Call Stack*).
- Coding exercises with already written test suites provide direct feedback to students' effort at solving the exercises.

Software testing is an important topic, which unfortunately is often neglected in CS/SE programs [3]. Furthermore, many students consider software testing ''tedious'' and ''boring'' [3]. For example, ''*Students often view testing as a boring and unnecessary task, and education is usually focused on building software, not ensuring its quality*'' [4]. However, if students can see the practical benefits of software testing (e.g., in the form of already existing unit tests in an *AlgDat* course), they might be more willing to learn and practice software testing.

In this paper, we report our experience and lessons learned in taking over the *AlgDat* course at Kristiania University College (Oslo, Norway), and how we shifted its focus from a more CS emphasis on mathematics to a SE focus, in which the use of software testing plays a major role. At the end of the course, a survey was given to the students, asking specific questions related to the use of software testing in *AlgDat*. The results were overwhelmingly positive: most respondents found learning software testing useful, which made the studying and understanding of the *AlgDat*'s topics easier.

From such results, we can positively recommend the introduction of software testing when teaching *AlgDat*. To help lecturers and professors doing the same for their courses, we provide as inspiration all of our teaching material online, on the GitHub repository https://github.com/arcuri82/algorithms

II. Course Structure

The course is divided in 12 lessons, teaching the following topics:

1) Arrays, Lists, and Unit Tests
2) Generics, Stacks and Queues
3) Runtime Analysis and Sorting
4) Recursion and Test Driven Development

might want to add that test suites are often used simply for evaluation & learning the tools can easily be left for students to deal with

This work is funded by the Research Council of Norway (project on Evolutionary Enterprise Testing, grant agreement No 274385).

978-1-7281-1075-2/20/$31.00 ©2020 IEEE
DOI 10.1109/ICSTW50294.2020.00075

5) Tree Maps
6) Hash Maps and Sets
7) Iterators, Lambdas and Streams
8) Graphs
9) Text Search and Regular Expressions
10) Decision and Optimization Problems
11) Genetic Algorithms and Randomness
12) Data Compression

The topics are divided in two main groups: the first nine classes covering the foundations, needed by any student wishing to work as a software engineer, while the last three classes being more advanced topics.

Each class starts by covering the theory, with traditional lectures using Powerpoint slides, followed by in details walkthroughs of code implementations, including showing and running the unit tests. The programming language used for the examples is Java, with unit tests written in JUnit 5. As unit tests are used in *every single class*, the topic of unit testing is introduced already in the very first class. Debugging is taught in the second class, once the students get taught as well the details of the JVM memory model (needed to understand how to read the frame information on the Function-Call Stack).

In the fourth class, students get introduced to *Test Driven Development* (TDD) [5]. When learning something from someone, one would be wise to ask ''*Do you practice what you preach?*''. And, in this context, the answer would be a firm *no*. We are not advocates for TDD, and we do not use it when developing software (neither when we worked in industry [6], and neither in our research projects, e.g., [7]). We simply do not find TDD useful (we prefer a top-down approach to software development). However, TDD could be useful for junior developers. This is the reason why the students in our *AlgDat* course get introduced to TDD, and are asked to do some exercises to practice with it. However, TDD is covered only in one single class, and never mentioned again in the rest of the course. It is up to the students to use TDD if they like it, or forget about it if they do not.

there is something here ...

III. Exercises

Theory is important, but, in Software Engineering, practice is even more important. One of the best ways to learn software development is to actually get your ''hands dirty'' by writing software. This is challenging in *AlgDat*, because students need to get constant feedback on the software they develop. For example, if a student is asked to write a mobile application, s/he can check by him/her-self if the application does run, or if it is not even starting, or crashing after the first user interaction. But how can a student check if their implementation of a *LinkedList* is actually working fine?

To cope with this problem, we followed this approach: for each exercise in which a student is requested to write a Java class, we first provide an *interface* with documentation on what needs to be implemented. Then, we also provide an existing *abstract* test suite with tests implemented against such interface. To clarify such approach, a simple exercise example is shown in Figure 1, where the interface

```java
public interface ArrayUtils {
    /**
     * @param array
     * @return the minimum value in the array
     * @throws IllegalArgumentException if input is
     *         null or empty
     */
    int min(int[] array) throws
    IllegalArgumentException;
}
```

Fig. 1. Exercise example, where students need to implement such interface.

```java
public abstract class ArrayUtilsTestTemplate {

    protected abstract ArrayUtils getNewInstance();
    protected ArrayUtils utils;

    @BeforeEach
    public void init(){
        utils = getNewInstance();
    }

    @Test
    public void testEmptyArray(){
        assertThrows(IllegalArgumentException.class,
                () -> utils.min(null));
        assertThrows(IllegalArgumentException.class,
                () -> utils.min(new int[]{}));
    }

    @Test
    public void testMinPositive(){
        int[] array = {5,4,1,7};
        int res = utils.min(array);
        assertEquals(1, res);
    }

    @Test
    public void testMinNegative(){
        int[] array = {5,-4,1,-7,0};
        int res = utils.min(array);
        assertEquals(-7, res);
    }
}
```

Fig. 2. Abstract test suite for the interface from Figure 1.

```java
public class ArrayUtilsImpTest extends
    ArrayUtilsTestTemplate {

    @Override
    protected ArrayUtils getNewInstance() {
        return new ArrayUtilsImp();
    }
}
```

Fig. 3. Implementation of the abstract test suite from Figure 2, where the overridden abstract method `getNewInstance()` returns a concrete instance of the student's solution implementing the interface from Figure 1.

`ArrayUtils` needs to be instantiated with a method that finds the minimum value in an integer array. The students are also provided with the abstract test suite in Figure 2. In such class, notice the *abstract* method `getNewInstance()`. Once the students make their implementation in a class called `ArrayUtilsImp`, they get taught how to automatically test it by creating a concrete class `ArrayUtilsImpTest` extending the abstract `ArrayUtilsTestTemplate`, like the one shown in Figure 3, where the overriden method `getNewInstance()` returns a new instance of the student's class `ArrayUtilsImp`.

The code repository of our *AlgDat* course is divided into three Maven modules:

- `lessons`: including all the code used during the lectures.
- `exercises`: all the starting code needed for the exercises, like the interface `ArrayUtils` and the abstract test suite `ArrayUtilsTestTemplate` used in our previous example.
- `solutions`: the solutions to the exercises, e.g., the classes `ArrayUtilsImp` and `ArrayUtilsImpTest`.

Testing does not prove correctness. Furthermore, even if a student's solution is functionally correct, it could be inefficient or convoluted. And it could also be that an exercise is too difficult for a student. So, besides providing a test suite to check for errors, we also provide the solutions to all the exercises.

Notice that writing software starting from a documented interface with an already written test suite is as far as you can get from actual software development in practice. It is done only for didactic reasons. Students must see first-hand the benefits of software testing before being required to write their own test cases (which we start to do in some exercises much later in the course).

Besides learning the benefits of software testing, and getting direct feedback when putting into practice *AlgDat* concepts, such an approach is also useful to see in practice the concrete (pun intended) benefits of Object-Oriented concepts such as *interfaces* and *abstract classes*.

IV. Survey

AlgDat is a complex subject. As researcher in software testing, and former senior software engineer [6], we know how important software testing is. On the one hand, we want to teach our students the importance of software testing, in a fashion that should not be too tedious nor boring. On the other hand, we worried that introducing software testing would had been too overwhelming for students, which might already struggle to catch up with all the complex topics required to cover in an *AlgDat* course. Furthermore, software testing is usually not taught together with *AlgDat*. There are some exceptions (e.g., [8]), but many (if not all) books on *AlgDat* do not even mention software testing (e.g., [9] used in the course).

To get feedback on whether teaching software testing in *AlgDat* was a good idea, we ran an online survey with 13 questions, to be completed before the exam. Out of 57 students that took the exam in autumn 2019, unfortunately only 16 responded (i.e., 28%). In the remaining of this section, we will list such questions and show the results of the survey (where ST stands for Software Testing).

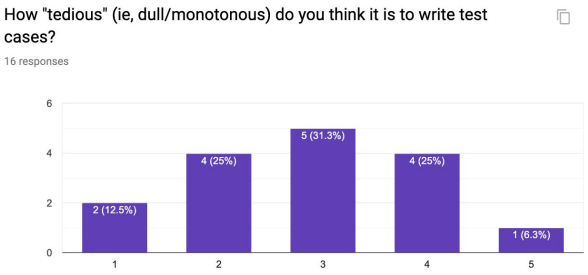

Fig. 4. Barchart from ''*Not at all tedious*'' to ''*Very tedious*''.

A. In which year of your degree are you in?

At Kristiania University College, BSc students in IT have a first common-year, and then the last 2 years are divided in 6 different specializations. *AlgDat* is a compulsory course in the 2nd year for 3 specializations: Programming, Game Programming and Intelligent Systems. However, *AlgDat* can also be chosen as elective in the 3rd year by the students in the other programs.

With this question, we wanted to see the composition of the students, as ''older'' students might have a different opinion compared to more junior ones, especially if they already seen some testing in other courses. Out of 16 students, 75% were 2nd year students, where the rest were 3rd year (with 1 student saying s/he was in the 4th year).

B. Taking into account your grades in previous courses, how would you consider your knowledge of coding (not testing) on average?

If you cannot write a for-loop, or even start an IDE, there is no much point in discussing more advanced topics like software testing. To avoid possible bias, in this question we asked the students to evaluate themselves based on their grades from previous courses (from failed `F` to outstanding `A`).

75% rated themselves with at least a `B`. The lowest score was a single `D`. Notice that only 16 out 57 students answered the survey, and the average of such reported scores is way much higher than the real average. This would seem to suggest that only the best and most motivated students bothered to answer the survey. The results of the survey need to interpreted keeping this information into account. copium, sorry to say

C. In how many courses have you seen/used ST before AlgDat?

For 37.5% of the students, this was the first time ever they saw software testing. 25% stated they saw in at least 1 other course, and another 25% stated they learned about software testing in 2 other courses. The other students saw it in 3 or more courses.

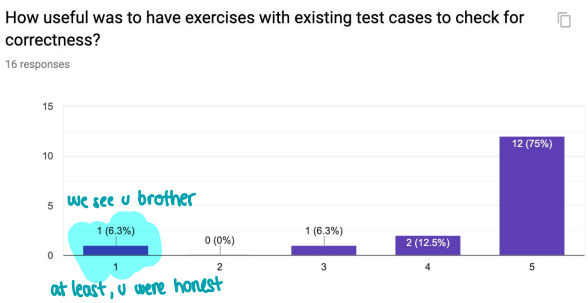

Fig. 5. Barchart from ''*Not useful at all*'' to ''*Very useful*''.

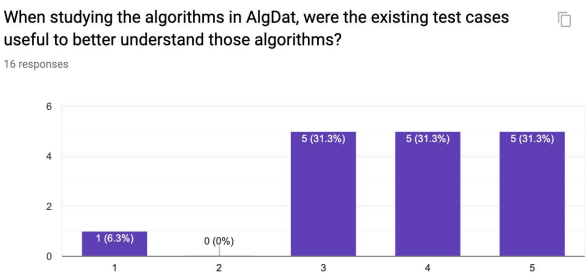

Fig. 6. Barchart from ''*No, not at all*'' to ''*Yes, very useful*''.

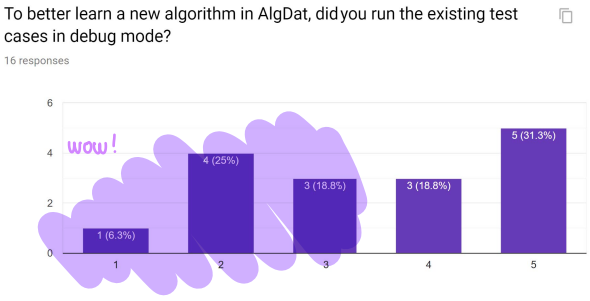

Fig. 7. Barchart from ''*Never*'' to ''*On most algorithms*''.

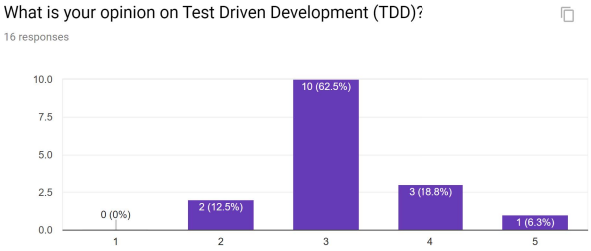

Fig. 8. Barchart from ''*Very negative*'' to ''*Very positive*''.

D. How ''tedious'' (ie, dull/monotonous) do you think it is to write test cases?

Figure 4 shows a normal distribution of the responses, centered around the ''neutral'' opinion. Considering how often software testing is reported as tedious and boring in the education literature [3], we found such results refreshingly surprising. We were expecting more students claiming that software testing is tedious. We can provide three hypotheses that might possibly explain such outcome:

1) Top performing students might better understand and appreciate the importance of software testing compared to students that have problems even in writing a for-loop. Recall that the sample of respondents to the survey is skewed toward top students (Section IV-B).
2) How we introduced and taught software testing did help in improving how students perceive it.
3) Only a smaller number of exercises required students to write test cases, and those were later in the course. And software testing was not a topic required for the exam. It might well be that some students did not do such exercises, and so did not really have any strong opinion on such question. In retrospective, we should have added a question to check if the students did do such kind of exercises. ya think?

E. How useful was to have exercises with existing test cases to check for correctness?

As can be seen in Figure 5, 75% of the students chose the highest score, which is *Very useful*. Having exercises like the ones we discussed in Section III is something that we can highly recommend, as very well received by students.

F. When studying the algorithms in AlgDat, were the existing test cases useful to better understand those algorithms?

In general, the students found the use of existing tests as good complementary material to better understand the code of the taught algorithms and data structures. However, as we can see in Figure 6, around 31% of the students were neutral about it.

G. To better learn a new algorithm in AlgDat, did you run the existing test cases in debug mode?

Our hypothesis is that students would have an easier time at understanding complex algorithms if they can run tests in *debug* mode, executing one step of the algorithms at a time. How to do it was shown in class during the lectures when discussing the code of lecture's topic. We hypothesized that then the students would do the same when studying such algorithms at home.

The survey provided mixed results on such hypothesis (see Figure 7). Although 31% of the students claimed to run the tests in debug mode on most algorithms, 25% stated they did it seldom. In retrospective, we should have asked more questions to understand why this was the case. For example, did the students find debugging difficult, and needed more resources to learn how to properly do it on their own? Or was rather that the slides/code/book were already enough to learn the topics of *AlgDat*?

H. What is your opinion on Test Driven Development (TDD)?

As discussed in Section II, we are no TDD advocate. Students get introduced to TDD in a single class, and never taught again, nor required to use it. It is no so surprising that 62% of the students have a ''neutral'' opinion about TDD (Figure 8), as likely they did not try to use it (e.g., they skipped the exercises

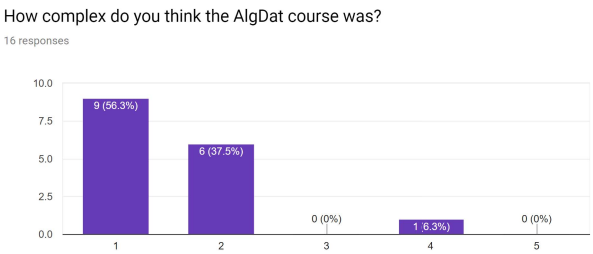

Fig. 9. Barchart from ''*Very complex*'' to ''*Very simple*''.

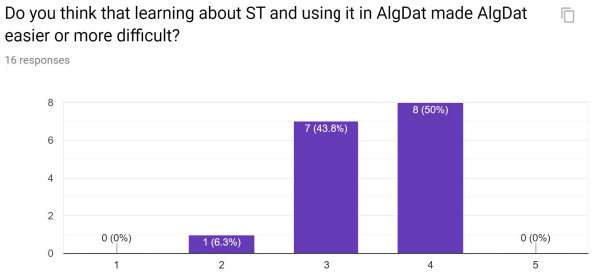

Fig. 10. Barchart from ''*Much more difficult*'' to ''*Much easier*''.

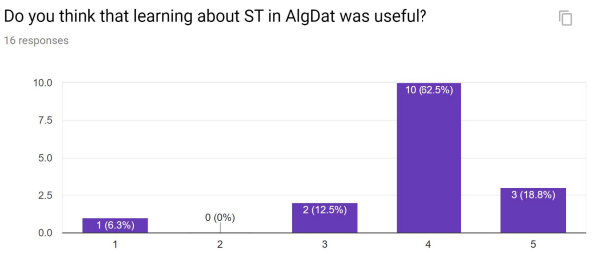

Fig. 11. Barchart from ''*Not at all*'' to ''*Yes, very useful*''.

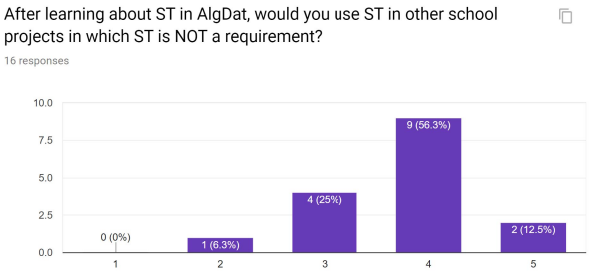

Fig. 12. Barchart from ''*No, likely never, unless I m forced*'' to ''*Yes, likely in all/most projects*''.

about TDD). However, it seems there are more positive than negative opinions regarding TDD.

In retrospective, we do not regret spending time in introducing the students to TDD. But we ponder whether, to make it a successful experience, we should spend more time on TDD in class, or simply remove TDD altogether and replace it with some other software testing topic (e.g., *Mutation Testing* [10]).

I. How complex do you think the AlgDat course was?

Not surprisingly, most students thought that *AlgDat* is a very complex subject (Figure 9). We asked this question to confirm if this was indeed what the students thought. Introducing a new, extra topic on an already challenging course might not be a wise idea. The load on the students could be just too much. However, our hypothesis is that software testing might help in making *AlgDat* more accessible and easier to study. This hypothesis is investigated in the next question.

J. Do you think that learning about ST and using it in AlgDat made AlgDat easier or more difficult?

To our relief, 50% of students thought that software testing helped them in better understanding *AlgDat* (Figure 10). Only a single student thought that learning software testing made the course *AlgDat* more difficult. We consider this outcome a *success*.

K. Do you think that learning about ST in AlgDat was useful?

Software testing is an important topic, regardless of *AlgDat*. While taking the *AlgDat* course, students are enrolled into other courses as well, in parallel. If maybe software testing is not so important for *AlgDat*, would the students still think that learning about it in *AlgDat* was not a waste of time? In Figure 11, we can see an encouraging 81% of students claiming it was good to learn about software testing in this course.

L. After learning about ST in AlgDat, would you use ST in other school projects in which ST is NOT a requirement?

Software engineering should not been learned in silos. During the progress of a BSc degree in IT, the skills learned in a course should be usable and useful for the following courses as well. Software testing should not be something learned in a single course in the second year, and then forgotten. If indeed software testing is a useful skill, students should decide to use it (e.g., in code projects) even when they are not required for their grades.

At this point, we cannot know if indeed the students will use their new software testing skills in other courses as well. However, based on the results of the survey (Figure 12), it would seem at least 68% of the students are willing to. This is definitively encouraging.

M. Do you have any further comment (in English) about the use of ST in AlgDat?

This last question had an open-text answer. Unfortunately, all students left it empty. ...they kinda... 6+2

V. Discussion

Based on the results of the survey, we can safely claim that introducing software testing in an *AlgDat* course is a good idea. We therefore encourage other *AlgDat* lecturers to do the same in their courses.

The main lesson learned is that students are very happy about having existing test cases to check the correctness of their exercise solutions. Not only it helps them to better understand and practice the concepts of *AlgDat*, but it is also useful to learn and appreciate the importance of software testing.

To help other lecturers to introduce software testing in their *AlgDat* courses, all of our teaching material is freely available online on GitHub (https://github.com/arcuri82/algorithms). Such repository can be used as starting point for inspiration.

For the future, it is important to consider what could be done to improve the course even further. We are currently considering the following:

- Add more exercises, with test cases and solutions.
- Investigate whether to introduce the students to *Mutation Testing* [10] as well, or if this should be done in a following course.
- Spend more time (and prepare more teaching material, including exercises) on the important topic of *debugging*.

VI. Related Work

Recently, we have carried out a systematic literature review on the topic of *Software Testing Education* [3], analyzing 30 academic articles published between 2013 and 2017. Throughout the years, many researchers and educators have shared their experience regarding how to teach software testing in CS/SE curricula.

Some authors describe different pedagogical approaches for teaching software testing, such as based on Vygotsky's zone of proximal development [11], Blooms Taxonomy [12], Flipped-Classroom [13] and CDIO [14], [15]. Other authors report their lessons learned, suggesting the use of real-world open-source projects in their teaching [16], [17], or encouraging pair testing [18]. *Gamification* seems a promising approach to engage students in the learning of software testing, e.g., with tools like *Code Defenders* [19].

probs check these?

VII. Conclusion

In this paper, we have discussed our experience in introducing the teaching of software testing in an Algorithms and Data Structures (*AlgDat*) course. We reported on lessons learned and the results of a student survey. It was a positive experience, where students appreciated being introduced to software testing. To help other lecturers to introduce software testing in their *AlgDat* courses, all of our teaching material is freely available online on GitHub (https://github.com/arcuri82/algorithms).

In our *AlgDat* course, students are introduced only to *Unit Testing*. In our following courses on web and enterprise development (e.g., material on GitHub as well, in the repository *arcuri82/testing_security_development_enterprise_systems*), they also learn about *Continuous Integration* and *End-to-End System Testing* using libraries such as *Selenium* and *RestAssured*. We will carry out a similar kind of student survey and report our lessons learned once those courses will be over in the next academic semesters.

References

[1] J. Offutt, ''Putting the engineering into software engineering education,'' *IEEE software*, vol. 30, no. 1, pp. 96--96, 2013.

[2] P. Ammann and J. Offutt, *Introduction to software testing*. Cambridge University Press, 2016.

[3] P. Lauvås and A. Arcuri, ''Recent trends in software testing education: A systematic literature review,'' in *UDIT (The Norwegian Conference on Didactics in IT education)*, 2018.

[4] D. E. Krutz, S. A. Malachowsky, and T. Reichlmayr, ''Using a real world project in a software testing course,'' in *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 2014, pp. 49--54.

[5] Y. Rafique and V. B. Mišić, ''The effects of test-driven development on external quality and productivity: A meta-analysis,'' *IEEE Transactions on Software Engineering*, vol. 39, no. 6, pp. 835--856, 2012.

[6] A. Arcuri, ''An experience report on applying software testing academic results in industry: we need usable automated test generation,'' *Empirical Software Engineering (EMSE)*, pp. 1--23, 2018.

[7] ------, ''EvoMaster: Evolutionary Multi-context Automated System Test Generation,'' in *IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 2018, pp. 394--397.

[8] I. A. Buckley and W. S. Buckley, ''Teaching software testing using data structures,'' *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, vol. 8, no. 4, pp. 1--4, 2017.

[9] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Addison-Wesley Professional, 2011.

[10] Y. Jia and M. Harman, ''An analysis and survey of the development of mutation testing,'' *IEEE Transactions on Software Engineering (TSE)*, vol. 37, no. 5, pp. 649--678, 2011.

[11] M. Allison and S. F. Joo, ''An adaptive delivery strategy for teaching software testing and maintenance,'' in *Computer Science & Education (ICCSE), 2015 10th International Conference on*. IEEE, 2015, pp. 237--242.

[12] R. Aziz, ''Improving high order thinking skills in software testing course,'' *International Journal of Computer Science and Information Security*, vol. 14, no. 8, p. 966, 2016.

[13] J. van Eijck, V. Zaytsev *et al.*, ''Flipped graduate classroom in a haskell-based software testing course,'' in *Pre-proceedings of the Third International Workshop on Trends in Functional Programming in Education (TFPIE 2014)*, 2014.

[14] Z. Bin and Z. Shiming, ''Experiment teaching reform for software testing course based on cdio,'' in *Computer Science & Education (ICCSE), 2014 9th International Conference on*. IEEE, 2014, pp. 488--491.

[15] S. Jia and C. Yang, ''Teaching software testing based on cdio,'' *World Transactions on Engineering and Technology Education*, vol. 11, no. 4, 2013.

[16] D. E. Krutz, S. A. Malachowsky, and T. Reichlmayr, ''Using a real world project in a software testing course,'' in *Proceedings of the 45th ACM technical symposium on Computer science education*. ACM, 2014, pp. 49--54.

[17] Z. Chen, A. Memon, and B. Luo, ''Combining research and education of software testing: a preliminary study,'' in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. ACM, 2014, pp. 1179--1180.

[18] I. Alazzam and M. Akour, ''Improving software testing course experience with pair testing pattern,'' *International Journal of Teaching and Case Studies*, vol. 6, no. 3, pp. 244--250, 2015.

[19] B. S. Clegg, J. M. Rojas, and G. Fraser, ''Teaching software testing concepts using a mutation testing game,'' in *Software Engineering: Software Engineering Education and Training Track (ICSE-SEET), 2017 IEEE/ACM 39th International Conference on*. IEEE, 2017, pp. 33--36.