From Chicago to London, a Proposal for Teaching Test-Driven Development

Nicolas Paez$^{1[0000-0002-0453-4259]}$

Universidad Nacional de Tres de Febrero, Saenz Peña, Argentina
nicopaez@computer.org

Abstract. Test-Driven Development is a very popular software development technique that offers well known benefits, but curiously it has a very low usage rate in the industry. Some authors suggest that this phenomenon can be due to a lack of knowledge and training. Aligned with this situation this article presents an approach to teach Test-Driven Development. The presented approach mixes Chicago and London Test-Driven Development styles and complements them with other Extreme Programming techniques. The described approach has been validated with positive results in two Software Engineering courses.

Keywords: Test-Driven Development · Education · XP

1 Introduction

Test-Driven Development (TDD) is a software development technique proposed by Kent Beck in the late '90. It was later formalized in 2003 when he published his book "Test-Driven Development By Example". This technique grew in popularity hand in hand with Agile Software development methods and in particular the method known as "Extreme Programming", also authored by Kent Beck. The popularity of this technique caught the attention of the academia and TDD was included in the computer science and information technology programs [1]. At the same time TDD has been the subject of several research projects and publications.

The benefits of TDD have been widely discussed in both formal and informal terms [2, 3]. However, the use of TDD in the industry is still very marginal, the most optimistic publications report a usage rate lower than 30% [4, 5].

As some authors have suggested [6], this situation may be due to a lack of knowledge and/or training on this technique. Aligned with this, the current article presents a TDD teaching approach based on the combination of TDD styles and some other Extreme Programming techniques.

2 The TDD Styles

As the name suggests, TDD proposes to guide the software development process by using automated tests following three steps:

1. Red: write a test and execute it to see it fail.
2. Green: write the minimum code to make the test pass.
3. Refactor: review the written code to ensure its clarity and modify it if needed.

These three steps are repeated until all requirements are met. Like Kent Beck explained [7], the technique can be applied at different abstraction levels, which means that we can write ”small/unit” tests or ”large/acceptance” tests. These different levels of abstraction lead to different styles of TDD. There are two very popular styles that are usually referred as Chicago and London, because of the software development communities that promoted them. These two styles became very popular after the publication of a series of videos by Bob Martin and Sandro Mancuso [8]. In these videos Bob (Chicago) and Sandro (London) show and compare the development of an application using these two styles.

In order to understand the differences between these styles we need to have in mind the Hexagonal Architecture Pattern which propose an explicit separation of concerns between the application core logic (in the center of the hexagon) and the adapters that connect it with the ”outside world” (in the borders of the hexagon).

The Chicago style, also known as ”Classic”, proposes an inside-out development flow. It starts developing the business logic components and then connect them with the outside world by adding interface and adapter components. This means that we start guiding the development with ”small/low-level” tests and then we end with ”acceptance/high-level” tests.

The London style, also known as ”mockist”, because it uses mock objects, proposes an outside-in development flow. It starts the development from the outside world which is typically the user perspective and then, it dives into the internals of the application. This means that we start guiding the development with ”acceptance/high-level” tests and then we end with ”small/low-level” tests. This style is very well documented in the book by Freeman and Pryce [9].

There are many publications about TDD, but just a few of them are focused on TDD education/training and at January 2022 none of them were focused on TDD styles.

3 The Proposed Approach

The proposed approach consists of starting by teaching TDD with the Chicago style (inside-out) and solving small problems/applications which do not require an external interface (or if they require one, it is a very simple one). This way, there is no need to write high-level tests. Then, once the fundamentals of the technique have been understood, moving on to study the London approach (outside-in) and using more complex/larger applications/problems. At this point we should also introduce mocking concepts. At the same time, this approach complement the London TDD Style with with the usage of other Extreme Programming techniques, since, as Kent Beck himself explains, the different Extreme Programming practices complement each other. Particularly important for this approach are the techniques of Continuous Integration, Pair Programming, and User Stories.

The idea behind this proposal is that Chicago style allow us to focus in first place in the implementation of the core components that encapsulate the business logic that according to the hexagonal architecture should not depend on the outside adapter components and its accidental complexities. This should make it easier for the students to direct their attention to TDD dynamics. Once the students dominate the TDD fundamentals, we can introduce the London approach which fits very well in the overall Agile development process. In particular, when requirements are modeled with User Stories, the acceptance criteria can be easy translated to "acceptance/high-level" tests that can trigger the outside-in TDD flow. This way the students can see the Agile Process in action with a detail guidance from requirements to code and tests. Of course that it is also possible to work in an agile way with a classical TDD approach, but in that case the students need more experience because the gap between "User Stories/Acceptance Criteria" and the low-level tests may not be easy to fill for novice developers. At the same time, Pair-Programming can help students to overcome technical difficulties while Continuous Integration gives them constant feedback and ensures that their code works also outside their own computers.

4 Validation and Results

This teaching approach has been used in the School of Engineering at University of Buenos Aires and also at National University of Tres de Febrero. In both cases, students have a first contact with Test-Driven Development in Programming courses of the first two years using a Chicago (inside-out) style. In these courses the students learn TDD and Object-Oriented Programming. They apply these techniques/concepts to solve small exercises/applications (without using a database or a web server). Then, in a Software Engineering course (1 or 2 years later) they return to TDD but with an outside-in style and they use it to solve larger exercises/applications. This Software Engineering Course is, in both institutions, in charge of the author of this article.

It is observed that at the beginning of this Software Engineering course the students understand TDD fundamentals and are able to apply it. But in general, they are not convinced to use it. If they can choose, most of them prefer not to use it and just use a Test-Last approach. During the Software Engineering course, students learn the outside-in TDD style along with the complementary Extreme Programming techniques mentioned in the previous section. In the final part of the course the students use all the studied techniques together in the development of a team project.

This teaching approach has been used since 2017, but it was not until 2022 that we decided to make a formal evaluation of it. Thus, once the second semester of 2022 was over, the students of the Software Engineering course in both universities were surveyed. They were asked on whether they would use the development approach studied. Specifically, the question asked was about the level of agreement with the statement: "I would use the development methodology studied in my future projects" and the possible answers were: a) totally agree, b) somewhat agree, c) neutral, d) somewhat in disagree, 3) totally disagree. The results showed that 58.3 % answered "Totally agree" while 35.5 % answered "Somewhat agree". It should be noted that all these students had passed the course, which implies that they demonstrated knowledge of TDD and the complementary techniques. The positive responses to the survey suggest that in addition to learning the techniques, the students consider the software development method studied to be convenient.

5 Conclusions and Future work

The experience described above indicates that start teaching TDD foundations by using the Chicago style and then teach the London style with complementary techniques and bigger problems/applications is effective. However, there are still some aspects of the proposal that should be evaluated, such as the set of tools and exercises used in the course. At the same time, there are implementation details of this proposal that were not covered in this article but that are relevant and that will be part of future publications.

Acknowledgments Some of the ideas presented in this article were refined in talks with my colleges Diego Fontevila, Alejandra Zangara and Carlos Fontela.

References

1. Paez, N.: Enseñanza de Métdos Ágiles de Desarrollo de Software en Argentina. Trabajo de Especialización. Universidad Nacional de La Plata (2020)
2. Khanam, Z. and Ahsan, M.N.: Evaluating the effectiveness of test driven development: Advantages and pitfalls. International Journal of Applied Engineering Research, 12(18), pp.7705-7716. 2017
3. Bissi, W., Serra Seca N., Adolfo G., Emer, M.: The effects of test driven development on internal quality, external quality and productivity: A systematic review. Information and Software Technology. **2**(74), 45–54. (2016)
4. Annual State of Agile Report. Last accessed 8 May 2023
5. Paez, N., Fontdevila, D., Gainey, F., Oliveros, A.:, F.: Technical and Organizational Agile Practices: A Latin-American Survey. Agile Processes in Software Engineering and Extreme Programming. Springer International Publishing (2018)
6. Causevic, A., Sundmark, D., Punnekkat, S.: Factors limiting industrial adoption of test driven development: A systematic review. Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation, ICST (2011)
7. Software Engineering Radio Episode 167: The History of JUnit and the Future of Testing with Kent Beck https://www.se-radio.net/2010/09/episode-167-the-history-of-junit-and-the-future-of-testing-with-kent-beck/. Last accessed 9 May 2023
8. Comparative Case Study: London vs. Chicago, https://bit.ly/3W1GtRl. Last accessed 8 May 2023
9. Freeman, S. Pryce, N.: Growing Object-Oriented Software, Guided by Tests. Pearson Education (1999)