Test-Driven Development Over the Past Decade: A Systematic Analysis and Survey Review

Nick Vicente Roland Kriesing Garcia

01.07.2024

Abstract

Test-Driven Development (TDD) is a valuable tool of modern software engineering, known for improving code quality and reducing defects. This paper examines its evolution over the past decade, exploring TDD's principles, traditional methodologies and newer approaches. A literature review underscores TDD's consistent enhancement of software quality but notes potential productivity impacts. Survey results highlight TDD's widespread adoption among the asked developers, citing benefits such as better code quality and faster bug detection. The literature and survey underscore minimal changes in TDD's core principles but reveal a diverse array of new approaches and tools. Future directions include AI integration and evolving testing frameworks, suggesting ongoing relevance despite debates on its applicability in agile environments.

1 Introduction

Test-Driven Development (TDD) has emerged as a cornerstone of modern software engineering practices, offering benefits such as improved code quality, enhanced branch coverage, and reduced defects. As software systems become increasingly complex, the need to maintain high standards of reliability and maintainability becomes crucial. This can be achieved through effective testing methodologies like TDD. Understanding the evolution of TDD over the past decade is important to adapt its practices to current software development effectively.

Despite its proven advantages, TDD faces resistance in adoption among development teams and challenges related to misconceptions about its benefits, maybe even in the evolution of TDD itself. Therefore, exploring the evolution of TDD practices and methodologies over the past decade, including advancements in its trajectory within software development, becomes imperative.

These challenges will be addressed and the evolution of TDD explained by reviewing foundational TDD principles, exploring traditional methodologies, comparing them to newer approaches, analyzing a conducted survey and evaluating the literature review and the survey.

2 Literature Review

2.1 Old/Common TDD Practices

TDD was first introduced over 20 years ago. Its principles originate from Kent Becks Book "Extreme Programming" published in 1999 and remain an integral part of TDD.

2.1.1 TDD advantages, disadvantages

Since the early 2000s, TDD was a well-established practice in software development. The adoption rate was on a moderate level [1] despite the major benefits that can emerge from using TDD. A systematic review analyzing empirical research on TDD's use in software development focusing on its impact on productivity and software quality was conducted by Bissi, Serra Seca Neto and Figueiredo Pereira Emer. They found out that TDD increases the software quality by over 75% but with a drawback of 44% decrease in productivity for the developing teams. Five additional studies support this tendency. The software and test quality tend to be improved, but in industrial settings there is decreased productivity and a high entry barrier. Writing effective unit tests is often cited as one of the most challenging aspects and TDD may not be suitable for all types of projects, as mentioned by Nanthaamornphong and Carver [2] [3] [4] [5] [6].

2.1.2 Traditional Chicago School and London Schools

The traditional approach to TDD is known as Chicago School or Style, focusing on strict adoption of the principles laid out by Kent Beck. These methodologies are for example the "Red-Green-Refactor Cycle", writing tests first, writing simple tests first, automated testing and avoiding over engineering as described in the book "Test Driven Development" by Kent Beck. This method is often referred to as Inside-Out, as tests are written from the inside, small unit tests, to the outside, end-to-end tests.

In the early 2000s another School of TDD arose. The London School of TDD was explained in a book written by Nat Pryce and Steve Freeman and became the second most important TDD practice. This TDD style employs an Outside-In approach, because the test are written the other way around, from the outside, focusing on the behavior the source code must have, to the inside, again small unit tests. In order to be able to test the behaviors, the London school of TDD uses mocks and stubs a lot, which adds complexity to the test process [7].

2.2 Current Practices and Methodologies

2.2.1 Appearance of new schools

Recently, new schools of TDD have emerged, further diversifying the landscape of TDD practices.

One notable example is the Munich School, which gained prominence in the early 2010s. The Munich School of TDD emphasizes a rigorous approach to software development, integrating formal methods and mathematical proofs to ensure the highest level of software correctness. By advocating for formal verification techniques and automated testing, this school aims to enhance both the reliability and correctness of software systems [8].

A new School of TDD is the Hamburg school. This school emphasizes a pragmatic approach to TDD, it's a hybrid between elements of behavior driven development (BDD) and state base testing, while highlighting the importance of Domain Driven Development (DDD) [9].

In contrast, the St. Pauli school of TDD is known for its emphasis on agility and flexibility. It promotes a lightweight and adaptive approach to TDD and values collaboration and communication within development teams, advocating for practices that enhance teamwork and facilitate knowledge sharing [10].

2.2.2 Other new methodologies

There is one more technique that emerged around 5 years ago and is called Test && Commit OR Revert (TCR). It mandates that any code changes must pass automated tests before being committed. If tests fail, the changes are automatically reverted, ensuring code integrity and promoting iterative, test-first development practices. TCR enhances code reliability by enforcing continuous validation, fostering a methodical approach to software development.

After discussing TDD schools and TCR it is useful to introduce the concept of testing shapes. The classical shape is the Testing Pyramid, which dictates many unit tests and fewer end-to-end tests. New shapes were introduced, such as the Ice Cream Cone or the Testing Trophy, adapting to the type of software that is being built.

2.2.3 Literature Table

When reviewing papers and other scientific literature on the principles and methodologies used in TDD, it became apparent that there was an insufficiency and lack of knowledge that could have informed this paper. Therefore, a meta-analysis was chosen to be conducted. A total of 116 books about TDD were identified through thorough reviews on the website goodreads.com. The following figure 1 illustrates the evolution of book publications and editions over time.

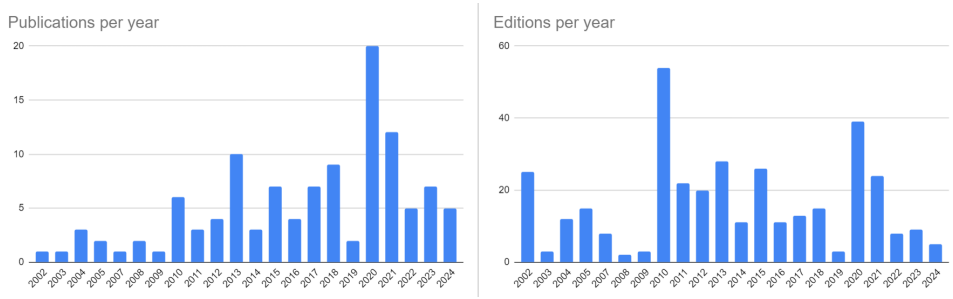

Figure 1: All TDD books: Publications and editions

This graph demonstrates a gradual but consistent increase in the number of books written about TDD in general, reflecting growing interest and adoption of TDD practices. From 2002 to around 2010, the number of books published annually on TDD was relatively low but increased gradually. This period likely represents the early adoption and exploration of TDD methodologies in software development. There is a significant uptick in the number of books published starting around 2010, with a peak in the mid-2010s (around 2013-2015). This surge likely reflects increasing interest, adoption, and maturation of TDD practices across various programming languages and domains. The trend

continues to show a generally upward trajectory with fluctuations. While the growth rate may have stabilized compared to the mid-2010s peak, the number of publications remains consistently higher than in earlier years.

Some books have multiple editions, like "Test Driven Development by Example" by Kent Beck (25 editions) or "Test-Driven Development with Python: Obey the Testing Goat: Using Django, Selenium, and JavaScript" by Harry Percival (26 editions) showing their popularity and continuous updates.

Many of these books are tailored to specific programming languages such as Java, C#, Python, PHP, and Ruby, among others. This trend suggests a widespread adoption of Test-Driven Development (TDD) across various fields and domains. However, it may not necessarily indicate a proportional increase in interest or the evolution of TDD principles themselves.

The following graph in figure 2 also shows the evolution of book publications and editions over the years, but only for books that are not language specific and talk about TDD principles or guides for applying TDD, totaling 25 books.

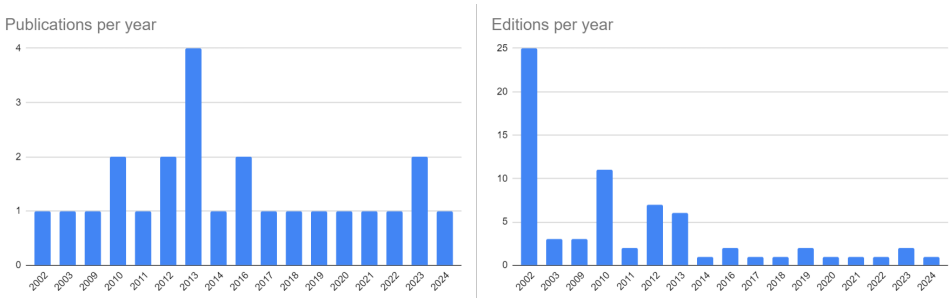

Figure 2: TDD principles books: Publications and editions

The table illustrates the slow progression of TDD literature about its principles over time, starting from Kent Beck's work in 2002 to recent publications in 2024. This evolution reflects the stagnating interest and development in TDD principles. However, the number of editions shows Kent Beck's book as the most enduringly popular, surpassing others' editions by far, with the last one being published in March 2022. The decrease in editions per year after the early peaks may suggest that the foundational TDD books have stabilized in content, requiring fewer updates.

3 Methodologies

3.1 Systematic Analysis Approach

Based on reviewing previously conducted studies and works the deduction of the change and use of TDD principles and methodologies was not feasible. Therefore, analyzing the impact on productivity and quality, the difference between older and newer methodologies and conducting a meta-analysis of TDD literature seemed comparable to a thorough empirical analysis.

Regarding the analysis of TDD's impact on productivity and quality in software development, as mentioned befor, multiple studies support the statement that code quality increases significantly while productivity decreases. The studied papers were evaluated through a table, examining each papers' working context, counting and weighing advantages and challenges with their application of TDD.

Subsequently, various existing TDD schools and their properties were mentioned and differences were explained. Here, the 5 selected schools are among the most popular schools of TDD with Chicago and London schools being the most used, since those two schools are the ones that are mentioned most of the time.

Later on a systematic meta-analysis of books about TDD, from the website goodreads.com, was performed. The books to be analyzed were chosen based on their titles, which contain "TDD" or "Test-Driven Development". All duplicates, being in different languages or editions, were removed. Due to the books being published over the course of time the evolution of the interest and change in TDD and its principles and methodologies can be seen. That is how the first two graphs came to existence. The second two graphs represent the books about TDD only concerning the principles and general application guides of TDD and no programming language specific literature.

3.2 Survey Design and Implementation

3.2.1 Key questions and data collection methods

To consolidate the findings of the literature review, a survey was conducted. Its goal is to gather how developers felt about TDD from a more personal angle than it is the case in literature reviews. The survey aimed to gather detailed insights on the benefits and challenges of TDD as experienced by developers, assess the evolution of TDD practices over the past decade, and understand the impact of emerging tools and AI on TDD with this set of questions:

In order to be able to interpret the answers to the survey, the background of the respondents was needed, to know their experience with TDD and in the software engineering world. To get those responses 3 specific question about their prior experience were asked. Furthermore, questions were asked to asses if TDD is used in their current team i.e. the teams acceptance and application of TDD, what the advantages and disadvantages of TDD are and to get if the how well TDD is integrated into the development process, including the use CI/CD and automated tests.

To get the information that is relevant for this study, the survey asked if the TTD principles, practices and tooling in TDD have evolved over the past decade, as well as identifying new tools and methods that have emerged in the past years. To further the horizon a question about the usage of AI in TDD was added and how TDD might or should change in the future.

3.2.2 Target demographic and bias

The survey design was finalized with the input from several respondents who helped modifying and add questions to create this optimal set of questions shown in table 1.

However, it is important to acknowledge potential biases in the study. The survey had 21 respondents, predominantly experienced programmers who are familiar with and generally favorable towards TDD. This could result in a skewed perspective that emphasizes the benefits of TDD while downplaying the challenges, given that these respondents are likely more proficient in TDD practices and may have a more positive outlook.

This demographic might not fully represent the wider software development community, which includes developers with varying levels of experience and differing views on TDD. Consequently, the survey results may not capture the full spectrum of opinions and experiences, particularly those of less experienced developers or those who have

Category	Questions
Experience	How much programming experience do you have?
TDD Usage	When did you first use TDD?
	In this timeframe, in how many projects did you use TDD?
Practices	Do you use CI/CD in your team or project?
	Do you have automated test runs in your CI/CD?
Acceptance and Application	How would you rate the overall acceptance and application of TDD in your current company or team?
Benefits and Challenges	What benefits have you observed from using TDD in your projects?
	What challenges have you faced when using TDD?
Evolution of TDD	Do you think TDD principles (like 'red, green, refactor', 'test first', etc.) have evolved over the past 10 years?
	How much do you believe TDD practices (methods, techniques, and approaches) have evolved over the past 10 years? (Slightly, Moderately, A lot)
Tools and Methods	Has the tooling for TDD become better?
	What tools and methods have emerged as a result of using TDD?
Impact of AI	Do you think AI has changed the application of TDD? If so, how?
Improvements Needed	What is missing from TDD as it is used today?

Table 1: Survey Questions

struggled with TDD. This bias could limit the study's ability to provide a balanced and comprehensive view of TDD's effectiveness and applicability across different contexts.

To mitigate this limitation and enhance the study's validity, future research should aim to broaden the participant pool to include developers across different experience levels and varying degrees of familiarity with TDD. This approach would provide a more representative and nuanced understanding of TDD's impact on software development practices and better inform discussions on its benefits, challenges, and future evolution.

4 Results

4.1 Survey Findings

To deduce the value of the respondents' answers, their experience and amount of use of TDD has to be assessed. Over 50% of respondents have more than 20 years of programming experience in general and in total, 90% have more than 10 years of experience. Almost 50% of respondents have used TDD for the first Time over 10 years ago and about 10% i.e. two developers have never used it. Of the people that have used TDD 72% use it in 50% of their projects or more. Furthermore 95% of the replies affirm that they use CI/CD in their current team and only 5% don't do automated testing in their process. These numbers indicate that a decent number of respondents have extensive experience in developing with TDD.

The question about the benefits observed from using TDD yielded the results in table 2.

The table indicates strong consensus among 19 respondents on the benefits of TDD. All respondents observed better code quality, and the majority reported fewer bugs, faster bug detection, improved maintainability, and higher developer productivity. The use of TDD definitely seems like it brings a lot of advantages to the developer, to the code base and everything around it. But also some challenges depicted in table 3.

The table reveals several challenges faced by software developers practicing TDD, including difficulties with legacy code, team resistance, and insufficient knowledge about TDD principles. Legacy code complexity often necessitates extensive refactoring to integrate TDD effectively, while team resistance highlights the methodical shift needed to

Respondent	Better code quality	Fewer bugs	Faster bug detection	Improved maintainability	Better documentation	Higher developer productivity
1	X	X	X	X	X	X
2	X	X	X	X		X
3	X	X	X	X	X	X
4	X	X		X	X	X
5	X		X	X		X
6	X	X		X		X
7	X	X	X	X		X
8	X	X	X	X		X
9	X	X	X	X	X	X
10		X	X	X	X	X
11	X	X	X	X	X	X
12	X	X	X	X	X	X
13	X	X	X	X	X	X
14	X	X	X	X	X	X
15	X	X	X	X	X	X
16	X	X	X	X		X
17	X	X	X	X	X	X
18	X	X	X	X		X
19	X	X	X	X	X	X

Table 2: Benefits observed from using TDD in projects

Respondent	Increased time investment	Difficulty with legacy code	Team resistance	Insufficient TDD knowledge	Tool support	Other Challenges
1	X	X	X	X		Hard to get right
2		X	X			
3			X	X		
4		X				
5		X	X	X		
6		X				
7		X	X			
8	X	X	X			
9					X	
10		X	X			
11	X	X	X	X		
12		X	X	X		
13		X	X			Deadlines kill TDD
14			X	X	X	
15				X		
16			X		X	
17		X	X			Unclear final code
18		X	X	X	X	
19		X	X			

Table 3: Challenges observed from using TDD in projects

embrace TDD's benefits. Additionally, challenges with tool support and the pressures of deadlines present practical barriers to consistent TDD implementation.

To gain further insights into the evolving nature of TDD, participants were also asked if they think that TDD principles changed over the past decade. This question aims to understand perceptions of how TDD principles have adapted to changing technological landscapes and development practices over the past decade. 38% of respondents say that TDD principles haven't changed at all and 43% that they have changed only a little bit. This indicates that a significant portion of practitioners perceive TDD principles as largely stable, with only minor adjustments over the years. This perception of stability might reflect the enduring relevance and effectiveness of core TDD principles in software development.

The next topics in question are TDD methods, techniques, approaches and tooling for TDD. The majority of respondents believe TDD practices have improved to varying degrees, with "Moderately Better" and "Slightly Better" being the most common responses.

Five respondents perceive significant improvements ("A lot Better"), while a minority, only two respondents, see negative changes ("Slightly Worse" or "A lot Worse"). This indicates a general consensus towards positive evolution, although with some exceptions. Such data highlights the mixed but largely favorable sentiment towards the advancements in TDD methodologies.

To add to this trend of advancements in TDD methodologies, a question about what tools and methods have emerged as a result of using TDD was answered by the respondents. Ten of the asked developers answered this question and four of those mentioned the practice TCR. Additionally, methodologies such as ATDD and BDD have also gained traction, further refining the scope and application of TDD principles. Many wrote about the tooling being easier to set up and covering more languages and use cases than before. New tools like jest, Playwright and Cypress have made testing more accessible and integrated into the development workflow, shifting the perception of testing from an afterthought to a fundamental part of the process.

Notably, the rise of code coverage tools and practices is mentioned as a counterpoint, suggesting that even the absence of strictly following TDD has driven innovations in testing tools. Respondents also emphasize the complexity of tracing these developments directly to TDD, although smaller steps in development and the adoption of methodologies like TCR are recognized as building upon the original TDD framework.

Looking at a specific tool, AI, some respondents are optimistic, noting that AI can automate the creation of tests and envision potential future tools where AI develops code implementations based on programmer-written tests. Out of 15 answers, 6 respondents claimed to have noticed no change in the application of TDD with the use of AI. A few respondents express doubt about the current capabilities of AI in this context, highlighting that TDD is fundamentally a design methodology rather than just a testing method, and suggesting that AI might only assist with more repetitive or mundane tasks for now. Overall, it appears to be that while AI has potential, it has not yet meaningfully transformed TDD practices.

Broadening the scope, the participants expressed their opinion on what is missing from TDD as it is used today. A recurring point is the lack of acceptance and understanding of TDD's long-term benefits, with many developers either not practicing it at all or only superficially. There is a call for better IDE support and tooling specifically made for TDD workflows. Another significant point is the perceived weakness in the adoption of TDD, which affects the quality of both test and production code.

4.2 Literature analysis

Empirical studies indicate exactly what was mentioned just before. TDD can significantly improve software quality, but often at the cost of reduced productivity. Traditional TDD practices, such as the Chicago School's "Inside-Out" approach and the London School's "Outside-In" methodology, have paved the way for newer methodologies like the Munich School, the Hamburg and St. Pauli Schools. Additionally, the Test && Commit OR Revert (TCR) methodology has emerged, promoting iterative, test-first development practices. A meta-analysis of TDD literature shows a consistent increase in publications and editions of books about TDD specific to a tool, language or similar, especially starting in 2010, indicating growing interest and adoption across various programming languages and domains. TDD's principles don't seem to have changed much; notable works like Beck's "Test Driven Development by Example" are frequently updated and widely ref-

erenced, but there are practically no other books of similar importance published since then.

4.3 Corelations between literature analysis and survey review

By examining both historical trends and contemporary perspectives, it becomes evident that while TDD principles have not evolved a lot, its methodologies and techniques do have matured and diversified through various TDD schools. The integration of AI into TDD processes presents a potential trajectory for future improvement, just as much as increasing the acceptance of TDD, understanding how it works and what benefits it brings to the table. The consistent call for better tools and deeper understanding highlights a gap between the established benefits of TDD and its practical implementation in modern software development.

5 Conclusion

The evolution of TDD can be divided into two parts. TDD principles, originating from works like Beck's "Test Driven Development by Example", have remained fundamentally unchanged over the past two decades. However methodologies, approaches and tools have evolved significantly. In the past decade, new TDD schools such as Munich, Hamburg and St.Pauli schools have emerged accompanied by advancements in testing frameworks and integration tools. These methodologies and tools reflect ongoing efforts to refine and adapt TDD principles to modern development practices, aiming to strike a balance between rigorous testing and agile delivery.

Some further thoughts about TDD raise the question if TDD is going to lose its relevance in modern software engineering, fueled by movements like "TDD is dead" [11]. This movement argues that TDD's test first approach may not always align with rapid prototyping and iterative development cycles favored in modern Agile practices. However, TDD's potential demise could be mitigated through a prominent advocate figure spreading TDD's benefits and abolishing the misconceptions many developers have about TDD. Maybe it would also not die if TDD methodologies were simplified and better tool support would make it more accessible across all developers.

The importance of continuously adapting TDD to the current software development landscapes is crucial. In order to propagate, consolidate and maximize its potential, TDD principles and methodologies requires everlasting education, experimentation and development.

References

[1] Tiago Silva da Silva Joelma Choma, Eduardo M. Guerra. Developers' initial perceptions on tdd practice: A thematic analysis with distinct domains and languages. `https://link.springer.com/chapter/10.1007/978-3-319-91602-6_5`, 2018. Accessed: 2024-06-20.

[2] Aziz Nanthaamornphong and Jeffrey C. Carver. Test-driven development in hpc science: A case study. *Computing in Science & Engineering*, 20:98–113, 2018.

[3] Wilson Bissi, Adolfo Gustavo Serra Seca Neto, and M. C. F. P. Emer. The effects of test driven development on internal quality, external quality and productivity: A systematic review. *Inf. Softw. Technol.*, 74:45–54, 2016.

[4] Aziz Nanthaamornphong. A case study: Test-driven development in a microscopy image-processing project. *2016 Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SE-HPCCSE)*, pages 9–16, 2016.

[5] B. Papis, K. Grochowski, Kamil Subzda, and K. Sijko. Experimental evaluation of test-driven development with interns working on a real industrial project. *IEEE Transactions on Software Engineering*, 48:1644–1664, 2022.

[6] Aziz Nanthaamornphong and Jeffrey C. Carver. Test-driven development in scientific software: a survey. *Software Quality Journal*, 25:343 – 372, 2015.

[7] Maciej Falski. Detroit and london schools of test-driven development. `https://blog.devgenius.io/detroit-and-london-schools-of-test-driven-development-3d2f8dca71e5`, 2020. Accessed: 2024-06-22.

[8] Marco Emrich. Testgetriebene entwicklung nach der münchner schule. `https://www.heise.de/hintergrund/Testgetriebene-Entwicklung-nach-der-Muenchner-Schule-6287450.html`, 2021. Accessed: 2024-06-22.

[9] Ralf Westphal. Hamburg style tdd. `https://ralfw.de/hamburg-style-tdd/`, 2019. Accessed: 2024-06-26.

[10] St. pauli school of tdd. `https://www.tddstpau.li/`. Accessed: 2024-06-26.

[11] David Heinemeier Hansson Kent Beck, Martin Fowler. Is tdd dead? `https://www.youtube.com/watch?v=z9quxZsLcfo&list=PLOGzxujsqdGDpW8mHsQwBByVYR2-9GX7u&index=1`, 2010.