Scientific Research Publishing

Journal of Software Engineering and Applications, 2025, 18(4), 139-147
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Test and Defect Driven Development (T3D): A Novel Approach to Software Development

Wasim Haque

Woodstock, GA, USA
Email: haque.wasim@gmail.com

How to cite this paper: Haque, W. (2025) Test and Defect Driven Development (T3D): A Novel Approach to Software Development. *Journal of Software Engineering and Applications*, **18**, 139-147.
https://doi.org/10.4236/jsea.2025.184009

Received: March 8, 2025
Accepted: April 18, 2025
Published: April 21, 2025

Abstract

Software development methodologies have evolved significantly, transitioning from traditional Waterfall models to more flexible Agile frameworks. However, the role of Quality Assurance (QA) in Agile methodologies has diminished, reducing QA's influence in the development process. This paper introduces Test and Defect Driven Development (T3D), a novel methodology that integrates QA more deeply into the software development lifecycle (SDLC). Unlike traditional Test-Driven Development (TDD), T3D emphasizes the proactive role of QA by creating and marking test cases as failed before development begins, allowing developers to fix defects in real time. This paper discusses the need for T3D, its advantages and disadvantages, and its potential impact on modern software development.

Keywords

Software Development, Software Engineering, Agile, Scrum, Development Methodology, TDD (Test Driven Development), QA (Quality Assurance), Defect Driven Development (DDD)

1. Introduction

Modern software development methodologies aim to improve efficiency, reduce defects, and enhance collaboration among teams. Agile methodologies such as Scrum, XP, TDD, and Defect-Driven Development (DDD) have gained popularity, but they often limit QA team's role just to execution rather than strategic involvement.

I'm proposing Test and Defect Driven Development (T3D) as an alternative that maintains Agile's adaptability while enhancing QA team's role. By having QA

define test cases before code implementation and pre-marking them as failed, T3D allows developers to address defects early. This paper explores the rationale behind T3D, its potential impact, and how it can integrate seamlessly into existing development frameworks.

Although T3D could work with Agile Teams of any size, irrespective of their geolocation and composition (Dev, QA and Ops). I think a perfect candidate for implementing T3D would be:

- In teams where QA is an integral part of their SDLC.
- In teams that have mature QA professionals with E2E knowledge of the product, underlying architecture, tools and processes.
- In small to medium sized product or project teams.

2. A Brief Look at Agile Software Development Methodologies

In the last decade, the adoption of Agile Methods has grown substantially, highlighting the need for a deeper exploration of their underlying principles. This paper offers a concise overview of some of the key frameworks currently in use and examines how they align with the core values outlined in the Manifesto for Agile Software Development [1].

2.1. Scrum

The Scrum development process addresses various technical and environmental factors that may evolve throughout a project. It is designed to help teams maintain their focus on software development while adapting to changing conditions [2]. Scrum is divided into three phases: pre-development, development, and post-development. The pre-development phase consists of two key components: planning and architecture/high-level design. During the planning stage, the team gathers system requirements and creates a list of features and modifications.

The architecture phase refines and evolves the design based on the backlog list. In the development phase, the team works in iterative cycles (sprints) to improve the system and introduce new functionalities. Each sprint typically includes tasks such as analysis, design, evolution, and delivery.

Sprints usually last between two weeks to one month. Typically, three to eight sprints are completed before the system is ready for release. However, there are some technology companies that release new or enhanced features in their software products every sprint. The post-development phase marks the conclusion of the development effort, during which no further modifications or features are added. Retrospective is also held at the end of the sprint to help all members share honest feedback on tasks, the project overall, and any challenges encountered. We then collaboratively brainstorm solutions to address these roadblocks, ensuring we're better equipped for upcoming sprints.

2.2. Extreme Programming (XP)

Like Scrum, Extreme Programming (XP) breaks down project work into short de-

velopment cycles known as "iterations" [3]. This approach allows Agile teams to quickly adapt to changing user requirements, even during the later stages of the product development lifecycle. Additionally, XP plays a significant role in enhancing product quality.

What distinguishes XP from other Agile methodologies is its strong emphasis on technical excellence specifically, producing high-quality code. Developers receive immediate feedback and continuously refine their code through practices like pair programming and both manual and automated testing. Project managers facilitate daily stand-up meetings to monitor progress and address obstacles. Meanwhile, product owners and stakeholders provide feedback based on acceptance tests and the product's performance after each iteration.

These core practices of XP foster effective teamwork, encourage adherence to coding best practices, and help maintain high standards for code quality and overall deliverables.

2.3. Test Driven Development

Test-Driven Development (TDD) is a software development methodology where developers write automated tests before writing the actual code [4]. This approach begins with creating unit test cases that define the expected behavior of a feature, guiding the subsequent development of the code itself. It follows an iterative cycle that integrates programming, unit test creation, and continuous refactoring.

The TDD workflow is structured around a repetitive cycle known as Red-Green-Refactor:

- Red Phase: The developer writes a test that describes a specific feature or behavior. Initially, this test fails because the corresponding functionality hasn't been implemented yet (hence, the "Red" phase).
- Green Phase: The developer then writes the minimal amount of code necessary to make the test pass, reaching the "Green" phase.
- Refactor Phase: Once the test passes, the code is refactored and optimized without altering its functionality, ensuring the test remains successful.

TDD helps maintain a reliable and bug-free codebase by catching errors early in the development process. It also promotes better design practices, as writing tests first forces developers to think critically about the intended functionality and structure of the code.

Additionally, since tests are an integral part of the development process, TDD leads to higher code coverage. It simplifies future modifications or refactoring by ensuring existing features remain stable and thoroughly tested.

Rooted in the principles of the Agile Manifesto and Extreme Programming (XP), TDD emphasizes test-driven development as the foundation for writing effective, maintainable code. Developers create small, focused test cases based on their initial understanding of each feature. Code is written or modified only when a test fails, reducing redundancy and ensuring that every piece of code serves a validated purpose.

2.4. Defect Driven Development (DDD)

Defect-Driven Development (DDD) is a software development approach where the process is heavily influenced by defects found during testing or production. Instead of following a traditional feature-driven or test-driven approach, teams prioritize fixing defects as they emerge, leading to continuous refinement and improvement of the codebase [5].

Key Characteristics of Defect-Driven Development:

- Defects as Requirements—Bugs and issues discovered during testing, user feedback, or production monitoring drive the development process.
- Iterative Fixes—Development focuses on resolving high-impact defects before adding new features.
- Continuous Testing & Validation—Frequent regression testing ensures that fixes do not introduce new defects.
- Shift-Right Approach—Emphasizes post-release monitoring, observability, and user feedback to identify real-world issues.
- Improves System Stability—Prioritizes reliability and user experience by addressing critical defects before introducing major changes.

Defect-Driven Development is often combined with Test-Driven Development (TDD) and Continuous Integration/Continuous Deployment (CI/CD) practices to create a balanced software quality approach.

3. Pitfalls of Agile Software Development Methodologies

While Agile, DevOps, and CI/CD have revolutionized software development by promoting speed and flexibility, they also introduce several potential pitfalls that teams need to be aware of to fully realize their benefits [6]. Top 3 potential pitfalls are discussed below.

3.1. Overemphasis on Speed over Quality

Rushing to meet tight deadlines can compromise code quality and lead to technical debt. For example, frequent sprints in Agile may pressure QA team to deliver quickly without sufficient testing leading to a potential customer escalation. In several cases, it has been seen that QA team is bullied into giving their signoff.

Poor software quality is incredibly costly for companies, both in direct financial losses and indirect impacts. Here are some key stats and insights:

- **$2.41 trillion**: The total estimated cost of poor software quality for U.S. companies in 2022 according to the Consortium for Information & Software Quality (CISQ).
- **$260 billion/year**: Spent on fixing software defects, including post-release bugs and maintenance.
- **30% - 40%** of a software project's total cost: Can be attributed to fixing defects introduced during development.
- Software failures caused losses of **over $1.7 trillion** globally in terms of outages, security breaches, and failed systems in recent years.

3.2. Lack of Clear Ownership

Lack of clear ownership can significantly hinder Agile development methodologies in several ways:

- **Ambiguity in Responsibility**—Without clear ownership, team members may not know who is accountable for specific tasks. For instance, who is responsible for testing the software and in charge of the quality, is it the QA team or the development team. This leads to delays in decision-making and unresolved issues, disrupting Agile's fast-paced, iterative nature.
- **Reduced Accountability**—Agile thrives on team ownership of deliverables. If roles aren't defined, accountability becomes diluted. This results in missed deadlines or incomplete tasks without any clear responsibility.
- **Bottlenecks in Collaboration**—Agile depends on close collaboration between developers, testers, and stakeholders. Without clear ownership, communication gaps increase, making it harder to address blockers efficiently.

3.3. Security Risks in Fast-Paced Development

Security checks are sometimes overlooked to maintain rapid deployment cycles. In DevOps, pushing code without thorough vulnerability scans increases the risk of breaches or in the absence of a QA team for security testing could potentially ship products to customers leaving them vulnerable to serious risks like DoS (Denial of Service) and SQL injections thus crippling their day-to-day operations leading to huge financial losses, latest one being the CrowdStrike security patch mayhem in the year 2024 that brought US Airlines operations to a halt resulting in loss of billions of dollars.

The data-driven reality highlights that modern Agile methodologies often prioritize velocity over quality, leading to a lack of motivation for delivering high-quality products. While Agile excels in speed, it frequently overlooks the importance of robust testing and defect management. So, how can we overcome these challenges while maintaining agility and ensuring QA teams regain their significance? One possible solution is a new methodology—T3D (Test & Defect Driven Development)—designed to address these shortcomings. Let's explore how it can enhance both quality and agility.

4. Test & Defect Driven Development (T3D)

You may come across information about DDD (Defect Driven Development) on the internet, but that's not the focus of this article. While it might sound like Test Driven Development (TDD), it's something entirely different—something you'll understand fully by the end of this article.

All development methodologies consist of multiple phases or steps which are interlinked with each other. It will be easier for the readers to understand if a walkthrough of those steps is given so that they can create an impression of the flow of process as it begins and ends in a chronological order:

1) The sprint begins with requirements or user stories being baselined and fi-

nalized after planning sessions.

2) Rather than the developers writing business logic first, the QA team prepares end-to-end (E2E) and Integration manual test cases, which are then reviewed by the Leads or Product Owners. Meanwhile in parallel, developers can complete their SCM duties, get architecture and design reviews, and begin writing unit tests.

a) For QA team to write effective manual E2E and integration TCs they should be fully aware of the functionality and its impact on other systems. This can be achieved by involving QA during sprint planning and grooming and other relevant design sessions.

b) The tests don't have to be very exhaustive but should be detailed enough for developers to execute and product managers to review.

3) Once the test cases are approved, the QA team marks the test cases as "Failed" in their test management tool.

4) Next, defects are raised for all failed test cases and assigned to the development team. Modern test management tools streamline this process by generating defects directly from test cases, reducing manual effort.

5) Developers then begin implementing business logic. As they complete features or functions, they conduct initial testing before releasing them to QA, checking for any defects identified in step 4 as their reference frame.

6) Development continues iteratively until all critical, high, and preferably medium-severity defects are resolved. As fixes are made, developers progressively release the updated codebase to QA.

7) Once QA receives the build, they retest the defects and take appropriate actions (Close or Re-Open). Resolved defects are marked as "Closed", and the corresponding test cases are updated to "Passed". Any test cases failing due to minor, low-priority defects remain marked as "Failed".

8) At this stage, Automation, Performance, and Security Engineers can begin their tasks, including automating E2E and integration tests, evaluating system performance, and conducting security scans to identify and mitigate vulnerabilities.

9) In parallel, testers can perform exploratory and regression testing to ensure no new defects have been introduced. Depending on severity, newly discovered defects may be fixed within the sprint, deferred to a future release, or moved to the next sprint. The QA team could also update the test suite with cases for any missed defects.

10) As the sprint nears completion, all critical and high-severity defects should be resolved. Automation, Performance, and Security Engineers finalize their tasks and provide feedback.

11) The sprint concludes with a review where all stakeholders (Developers, QA, and DevSecOps) showcase their work and determine whether to release the product or feature. If approved, the codebase is released to production; otherwise, unresolved issues are added to the next sprint's backlog.

12) Finally, a sprint retrospective allows teams to reflect on the process and identify areas for improvement, fostering continuous enhancement of workflow and efficiency.

The pictorial presentation of T3D can be seen below in Figure 1.

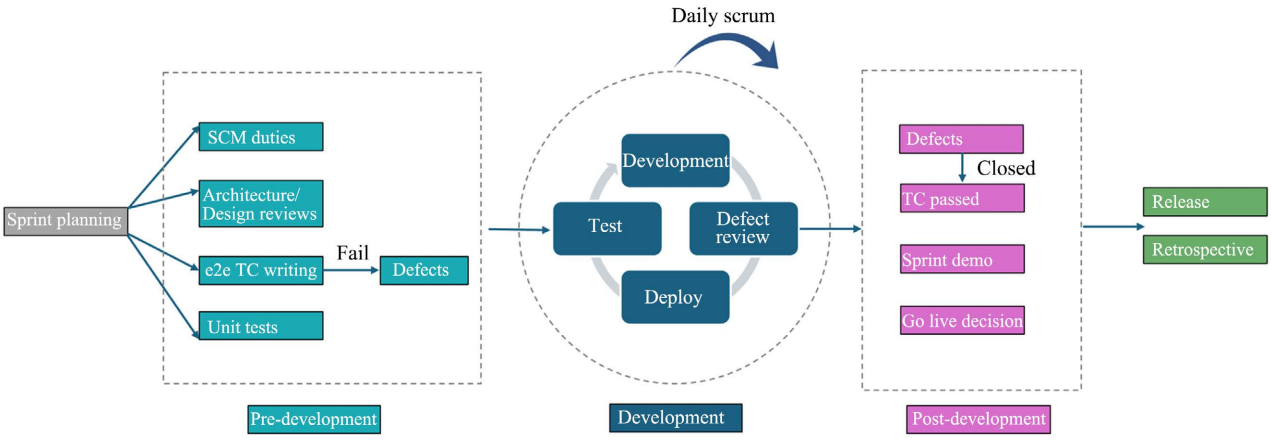

Figure 1. T3D development process.

Example Implementation

Let's understand the process better with the help of an example. Say an agile team is planning to work on 3 user stories of varying complexities.

Post sprint planning the QA team should come up with a map and get it reviewed by the Product and Dev teams as shown below in Table 1.

Table 1. User story map.

User story Id	Test & Defect		
	No. of E2E tests	No. of integration tests	No. of defects
US-001 (High)	3	5	8
US-002 (Medium)	2	3	5
US-003 (Low)	1	2	3

As developers write code and when believe they've logically concluded a function or a story they will begin with testing with 2 goals:

1) Acceptance criteria documented in the story is achieved.

2) The 16 defects raised above don't appear to block or impact the story.

This improves the build quality that will be delivered to the QA team, there won't be any surprises and the turnover and back-forth between Dev and QA will also reduce. As a result, Regression testing, Performance and Security evaluations would be faster.

In any iterative development process not all stories may be delivered to the QA team, there's a possibility stories delivered earlier may develop defects due to code changes done for other stories, but the chances of this happening in T3D are very less due to the nature of tests (E2E & integration) written by QA for devs. However, if by chance Devs miss anything QA is there to provide the safety net.

Once all the 16 defects (plus additional that team would have discovered) are

resolved and closed, the 16 test cases (6 E2E & 10 intg.) are now marked as "Passed". If the team decides to defer any defects, then the corresponding test case remains in failed state until the defect is resolved as shown below in Table 2.

Table 2. Sample results.

Tests	Test & Defect		
	Story Id	Test result	Defects open
TC-01	US-001	Passed	N/A
TC-05	US-002	Passed	N/A
TC-10	US-003	Failed	BUG-003-10

During sprint review, everything gets reviewed, and decisions are made to release features and to defer some to upcoming releases. Next sprint same process is followed where the team works on new backlog items along with multi-sprint stories and defects that were deferred from previous sprints.

5. Pros and Cons of T3D

Like any agile development methodology discussed in Section 2, T3D too has its advantages and disadvantages.

5.1. Pros

- **Early Defect Resolution**: Developers address defects before delivering a single build to QA, reducing bug-fix cycles.
- **Enhanced Collaboration**: QA team and developers interact more frequently, fostering better communication and teamwork.
- **Stable Builds**: Since defects are fixed early, regression testing is more effective, and builds are more reliable.
- **Increased QA Visibility**: QA plays a strategic role, defining quality benchmarks before development begins.
- **Improved Code Quality**: Developers have a clearer understanding of potential pitfalls and quality expectations from the outset.
- **Ease of Project Reporting**: Defect resolution progress is easily trackable by monitoring defect closure rates.

5.2. Cons

- **Similarity to TDD**: T3D resembles TDD but replaces unit tests with functional E2E and integration tests, which may create initial resistance.
- **Team Dynamics Shift**: Developers may initially resist the methodology due to the increased emphasis on defects.
- **Learning Curve**: Teams accustomed to traditional Agile or TDD may require training and adaptation time.

6. Discussion

T3D represents an evolution of test-driven methodologies, adapting to the increasing complexity of modern software systems. Unlike traditional TDD, which emphasizes unit testing, T3D integrates functional test cases as a precursor to development, ensuring real-world defects are anticipated and addressed proactively. The methodology does not conflict with existing Agile frameworks; instead, it enhances them by reinforcing collaboration and ensuring continuous quality.

By leveraging defect tracking tools and automation, T3D can be seamlessly incorporated into DevOps pipelines. Companies that adopt T3D may experience higher first-pass success rates, reduced testing cycles, and improved overall software quality.

7. Conclusion

T3D offers a balanced approach to software development, addressing the gaps left by traditional Agile and TDD methodologies. By involving QA at an earlier stage and ensuring developers have clear defect insights before coding, T3D enhances efficiency, collaboration, and product stability. While adoption challenges exist, organizations willing to experiment with T3D can benefit from improved software quality and a more cohesive development process.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] Agile Alliance (n.d.) Agile 101. https://www.agilealliance.org/agile101

[2] Schwaber, K. and Sutherland, J. (2020) The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. Scrum.org & Scrum Inc. https://www.scrumguides.org/scrum-guide.html

[3] Beck, K. and Andres, C. (2004) Extreme Programming Explained: Embrace Change. 2nd Edition, Addison-Wesley.

[4] Beck, K. (2002). Test-Driven Development by Example. Addison-Wesley.

[5] Nachiengmai, W., Ramingwong, S., Cosh, K., Ramingwong, L. and Eiamkanitchat, N. (2019) Defect-Driven Development: A New Software Development Model for Beginners. *Geomate Journal*, **17**, 149-155. https://geomatejournal.com/geomate/article/view/2150

[6] Shore, J. and Warden, S. (2007) The Art of Agile Development. O'Reilly Media.