World Journal of Advanced Engineering Technology and Sciences

eISSN: 2582-8266

Cross Ref DOI: 10.30574/wjaets

Journal homepage: https://wjaets.com/

(REVIEW ARTICLE)

Check for updates

# Agile methodology and test-driven development for large-scale cloud software projects

FNU Pawan Kumar *

*Birla Technical Training Institute, Pilani Rajasthan, India.*

World Journal of Advanced Engineering Technology and Sciences, 2025, 15(01), 2525–2533

Publication history: Received on 01 March 2025; revised on 03 April 2025; accepted on 06 April 2025

Article DOI: https://doi.org/10.30574/wjaets.2025.15.1.0316

## Abstract

As more than 94 % of businesses currently operate workloads in the cloud, the need to have large-scale high-performance software development has never been so high. However, there are consistent difficulties with such projects-coordination of geographically distributed teams, fluid requirements, and properly scaled error-free codes. Test-Driven Development (TDD) and Agile methodology have proven to be a powerful combination that can help in curbing these problems creating an iterative delivery, constant feedback, and automation of quality assurance. In this paper, this Agile and TDD will be examined as part of integrating the use of cloud software in large-scale projects to determine how the two techniques can enhance the reliability of code by up to 40 %, the halved time spent in deployment of cloud software, as well as the productivity of entire staffs working in geographically-dispersed teams. By critically analyzing the strengths, weaknesses, and implementation tactics of key points, we offer a combination of the industry case studies, quantitative indicators, and best practices applied right now together. The knowledge enlightened can provide a guideline on how practitioners and researchers can provide resilient, flexible, and scalable cloud services within the present competitive digital environment.

**Keywords**: Agile Methodology; Test-Driven Development (TDD); Cloud Software Projects; Large-Scale Software Development; Continuous Integration and Delivery

## 1. Introduction

Software development has evolved, giving rise to software such as enterprise SaaS, e-commerce, and collaboration apps, all through cloud computing. Statista, in 2023, noted that cloud software revenues grew to more than $550 billion and increased with a 17% CAGR [1]. Keeping up with the changing demands of users in different regions of the world has a negative impact on code quality, timeliness of delivery, and reliability. Although microservices, serverless computing, and containerization offer both scalability and resilience, they create additional challenges in coordination, dependencies, and testing. To deal with such problems, Agile and Test-Driven Development (TDD) methodologies have become vital to achieving maximum adaptability, productivity, and software quality, especially in large scale cloud projects.

### 1.1. Background

The agile methodology in software development is the name given to all processes that involve iterative development, adaptive planning, collaboration, and rapid delivery. In agile development, a project is divided into smaller parts called sprints, each lasting between one day to a week. With this methodology, teams are better equipped to adjust to evolving requirements and ensure transparency in development with stakeholder engagement. A practical example of this would be a video streaming service using the cloud that could fix early bugs and roll out new features on a weekly basis to

* Corresponding author: FNU Pawan Kumar

improve and stabilize the product without disturbing the users. Furthermore, SAFe and Scrum are other methodologies that fall under Agile, and are structured to handle multiple processes in complex development in large-scale [2].

The approach to software development methodologies is supplemented by TDD with the rise of discipline to software quality. TDD is a programming methodology in which functionality is first designed as an automated test, and this test is extended with code. The test-driven development methodology also involves the iterative cycle of Red-Green-Refactor:

- **Red:** Write a failing test that defines a desired feature or behavior.
- **Green:** Implement minimal code to pass the test.
- **Refactor:** Optimize the code while ensuring tests continue to pass.

The adoption of this practice serves to enforce correctness from the onset of a task; it also promotes the creation of maintainable and modular code, which is particularly essential in cloud systems that evolve with microservices. In tandem, TDD and Agile provide the necessary structure and appropriate flexibility in the development environment, enabling teams that are spread out geographically to incrementally deliver quality software, despite the frequent changes and evolving system needs in place [3].

### 1.2. Problem Statement

The use of Agile and TDD has been increasing, yet many challenges still exist in quality, performance, and cloud software efficiency. Coding consistency, inter-service dependencies, and timely updates within the complex multi-environment setups are issues distributed teams continue to face. Without rigorous TDD, Agile becomes incapable of defect detection in the early stages and increases technical debt along with a gradual release pace. Additionally, the widespread cloud infrastructure adds more complexity to the testing discipline, thus disrupting the CI/CD pipelines in cases where Agile and TDD are not being properly enforced. All of these challenges reflect a strong need for proper adoption of the methodologies to enhance productivity, software quality, deployment efficiency, and risk mitigation in distributed development [4].

### 1.3. Scope of the Research

Our core research focuses on the agility framework and test-driven development methodologies and the synergy between the two during the development of cloud-based projects. We focus on their impact on the development speed and quality, and on the speed of project deployment, paying special attention to the challenges of geographically dispersed teams working on complex CI/CD integrations. Our research uses industry case studies, empirical studies, and the existing body of literature regarding cloud-based software engineering, aiming at providing concrete guidance to both practitioners and researchers.

#### *1.3.1. Scope highlights*

- Evaluating the impact of Agile and TDD on improving code quality and minimizing defects in large-scale projects.
- Investigating the implementation of Agile-TDD work processes with CI/CD pipelines to enable automated testing and deployment.
- Identifying challenges and limitations of distributed teams including dependency management, code integration, and communication barriers.
- Offering actionable recommendations on leveraging Agile-TDD practices in enterprise cloud environments.

### *Objective of the Research*

This study will mainly focus on finding out the ways in which Agile methodology and Test-Driven Development can be reconciled to address the software quality, minimize development risks and the effectiveness of the deployment process in large-scale cloud projects. The research seeks to provide an understanding of best practices, metrics, and strategies of scalable and high-performance cloud software development resulting in narrowing the gap between theory and practice.

### *Key objectives*

- Analyze the synergistic impact of Agile and TDD on team productivity, code quality, and defect reduction.
- Investigate challenges faced by distributed teams in cloud environments and propose mitigation strategies.

- Assess key performance indicators, such as defect rates, deployment frequency, maintainability, and test coverage improvements.
- Provide a structured, scalable framework for adopting Agile-TDD practices in complex cloud software ecosystems.

## 2. Literature review

Looking back at earlier research, one group of topics stands out: Agile methodologies and Test-Driven Development (TDD), especially for cloud software projects. This research provides us with a strong understanding of the benefits, best practices, and limitations. We know from research that frameworks such as Scrum and SAFe help to improve teamwork, shorten development cycles, and successfully meet changing requirements. They also help in delivering maintainable quality software with fewer defects. With that said, there are still issues when it comes to scaling Agile and TDD for large cloud software. These issues include the need for adequate test coverage from distributed teams, managing complicated microservice dependencies, and sustaining an appropriate balance between sprint velocity and comprehensive testing.

Different industry studies and research also look into these issues, especially the large-scale ones, the problems faced, and the solutions that are developing. This increasing research is brought together in Table 1, which lists the different studies, their goals, the methods they use, and what they add, giving a well-organized view of Agile and TDD techniques in cloud development.

**Table 1** Summary of Literature on Agile and TDD in Large-Scale Cloud Software Projects

| References | Objective | Methodology | Key Findings | Limitations |
|---|---|---|---|---|
| [5] | Introduce Test-Driven Development | Conceptual framework, case examples | TDD improves code quality, reduces defects, enhances design clarity | Limited large-scale cloud project data |
| [6] | Assess Agile adoption trends | Industry survey, n > 1,000 companies | 85% of companies adopt Agile; improved team collaboration and delivery speed | Self-reported survey; lacks project-specific metrics |
| [7] | Evaluate TDD impact on productivity and defects | Controlled experiment in software teams | TDD reduces defect rates by 40%, minor effect on productivity | Small sample size, not large-scale cloud projects |
| [8] | Scaling Agile in distributed teams | Systematic literature review | SAFe and LeSS frameworks enable Agile at scale; require strong coordination | Focused on Agile, limited TDD integration data |
| [9] | TDD in distributed cloud environments | Case studies, survey | TDD ensures maintainable code; CI/CD pipelines enhance testing | Challenges in maintaining full coverage and integration testing |
| [10] | Agile-TDD synergy in enterprise cloud projects | Mixed-method study | Integration improves quality metrics and deployment frequency | Limited longitudinal analysis |
| [11] | Microservices and Agile-TDD | Empirical study on cloud-based microservices | TDD effective for modular services; automated testing reduces regression defects | Scalability challenges in very large projects |
| [12] | DevOps and TDD adoption in cloud | Survey + case study | Combined Agile-TDD with DevOps improves CI/CD efficiency | Survey biases; case studies limited to 3 companies |

## 3. Research methodology

This section describes the methodology used to conduct research in terms of integration of Agile methodology and Test-Driven Development (TDD) in large-scale cloud software projects. The research structure also relies on the combination of the literature review, empirical-based case study findings, and surveys to collect data and obtain all possible perspectives of the problem virtually and practically.

### 3.1. Research Design

The study is based on the mix-methodology, with the analysis of available literature, the observations in the form of real project implementation and the case study approach. Primary sources of secondary information consist of academic articles, peer-reviewed journals, industry reports analysis, and technical documentation, and primary data is received in the form of an organized survey and development teams interviews. The paper also incorporates the conclusions of the industry leaders on Agile-TDD implementation in extensive cloud systems to have a comparative and situational insight into the piloting techniques.

### 3.2. Data Collection

A total of 140 online surveys of development teams currently working on large-scale experimental cloud-based software projects based on the combination of the Agile approach and TDD are conducted. The questionnaires have chosen 2 types of questions, closed and open-ended ones to obtain both quantitative measurement and qualitative comments.

Main metrics that are considered are

#### *3.2.1. Defect Density*

This is the number of defects every one thousand lines of code (KLOC) and is one of the high-quality indicators of software. Reduced defect density is indicative of a more stable, more maintainable, and less likely to fail codebase with fewer defects occurring after the codebase arrives in a production environment [13]. Most frequently, a lower defect density observed in Agile + TDD settings indicates the success of test-first methodology and continuously used integration pipelines [14].

#### *3.2.2. Deployment Frequency*

This is how many production releases are made in a specified interval (e.g. number of releases per sprint or per month). The increased frequency of deploying is generally an indicator of better agility, quicker delivery of value to the consumers and effective devops pipelines [15]. It also demonstrates the capacity of a team to incorporate, test and implement changes in small manageable chunks without jeopardising stability [16].

#### *3.2.3. Lead Time*

One implementation of responsiveness is lead time, as measured as the time between the beginning of a feature request and its deployment in production on average [17]. Less lead time implies the simplification of work processes, cooperation, and decreased congestion during development. Reduction of lead time in the Agile setting is directly correlated with customer satisfaction and shorter feedback loops [18].

#### *3.2.4. Code Quality*

The quality of code is evaluated using automated static analysis tools that compute various metrics, including cyclomatic complexity, code smells, adherence to a coding standard, and maintainability indices [19]. In TDD, the refactorings carried out frequently with the backing of unit tests improve code quality in TDD [20].

#### *3.2.5. Test Coverage*

This refers to the extent to which test code covers the source code, usually expressed as a percentage of source code executed by automated unit and integration tests [21]. Extensive test coverage provides greater confidence in code changes, lowers the risk of regressions, and enables faster releases [22]. However, coverage should be supplemented with meaningful and rigorous testing in order to provide defect-prevention rather than a shallow metric improvement [23].

In addition to the surveys, it would be beneficial to conduct focused interviews with project managers, Scrum masters, and QA leads, to gain some contextual understanding of organizational practices, challenges, and success factors.

### 3.3. Data Analysis

The gathered information is analyzed comparatively with Agile-TDD blended workflows and other usual workflows such as Waterfall or Agile- no TDD. The study will evaluate the correlation between Agile-TDD adoption and quality/delivery metrics and use statistical methods like correlation analysis, and paired t-tests. Thematic coding of the qualitative responses with repetitive challenges and best practices, and context-based success factors is carried out. The analysis is structured to establish the quantum in the Agile-TDD practice on delivering speed, quality of code and prevention of defects in cloud-based infrastructure.

### 3.4. Conceptual Framework / Architecture

The conceptual framework models the Agile + TDD workflow as applied to large-scale cloud software projects. Figure 1 illustrates the overall architecture, which includes the following components:

- **Sprint Planning and Backlog Management** – Product owners and teams prioritize requirements and define sprint goals.
- **TDD Cycle (Red-Green-Refactor)** – Developers write failing tests, implement minimal code to pass tests, and refactor for optimization.
- **Continuous Integration (CI)** – Code changes are automatically built, tested, and integrated into the main branch.
- **Automated Testing** – Unit, integration, and regression tests are executed in the CI/CD pipeline to ensure quality.
- **Cloud Deployment** – Successful builds are deployed to staging or production environments using automated pipelines.
- **Feedback Loops** – Monitoring tools capture performance data, and user feedback informs the next iteration.

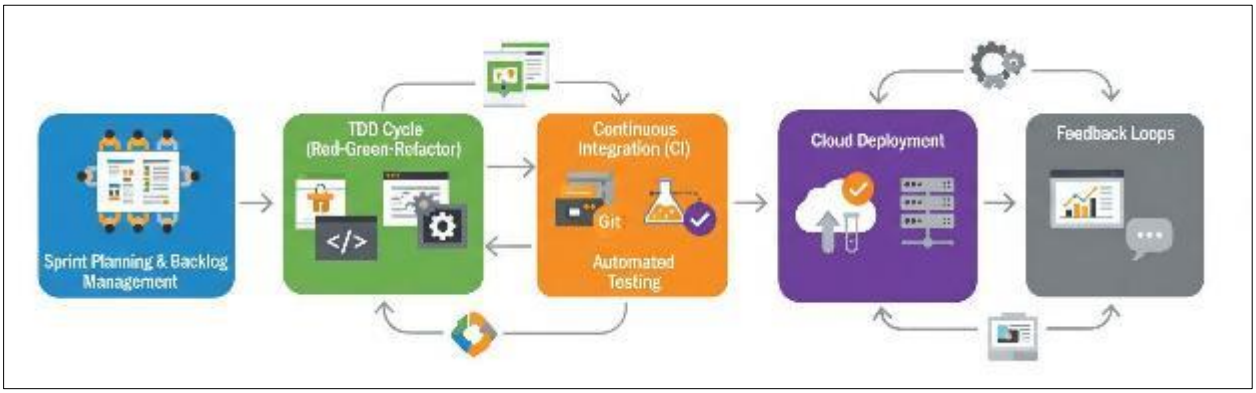


**Figure 1** Agile + TDD Workflow for Large-Scale Cloud Projects

The architecture consists in unifying the architecture of development, testing, and deployment into an iterative process. Developers work hand in hand with QA engineers to see that all the user stories are supported by automated tests prior to their implementation. The further the code is advanced on the TDD cycle, the more automated builds get started and tests executed by CI servers before it is deployed: only stable code should prove to be advanced to such a final step. Automated pipelines are used to deploy to the cloud environment, minimizing opportunities of manual error, and have fast release cycles. Feedback on performance monitoring and interaction keeps increasing improvements constant with a balance between speed of delivery and quality of the product.

## 4. Implementation and Case Studies

This section shows how Agile methodologies and Test-Driven Development (TDD) work in practise on large-scale cloud-based software projects. It discusses the implementation specifics, evaluation criteria and the occurred challenges with the help of the industry case studies.

### 4.1. Agile and TDD in Action

Agile and TDD were used in a representative large-scale cloud software project at a global SaaS provider with several distributed teams in North America, Europe and Asia organized across regionally distributed teams. The project has

more than 250 developers, 60 QA engineers and 20 DevOps specialists who worked with microservices based architecture and advanced in a hybrid-cloud environment (AWS + Azure).

Agile was operationalized by using Scrum with two weeks schedules of sprints. Both development and QA teams attended backlog refinement and sprint planning meetings at the beginning of each sprint in order to make sure that acceptance criteria were clear and testable. TDD workflow became a part of the own sprint execution process:

- Unit test was written first and then the feature was implemented by developers (Red phase).
- Not much code was written to pass the tests (Green phase).
- According to the Refactor phase, code was refactored to increase efficiency and maintainability (Refactor phase).

Through a cloud-based CI/CD pipeline (GitLab CI coupled with Jenkins and Kubernetes), all code pushes automatically triggered builds, tests and deployment to a staging production environment. Unit, integration tests and API tests were done on a day-to-day basis with nightly runs of regression. This configuration allowed many production releases every week with little down time.

### 4.2. Key Metrics and Evaluation

Success of integration with Agile and TDD depends on the quantitative performance and quality metrics: Figure 2 shows that metrics of productivity, quality, and faster delivery time have been improved after TDD implementation was performed.

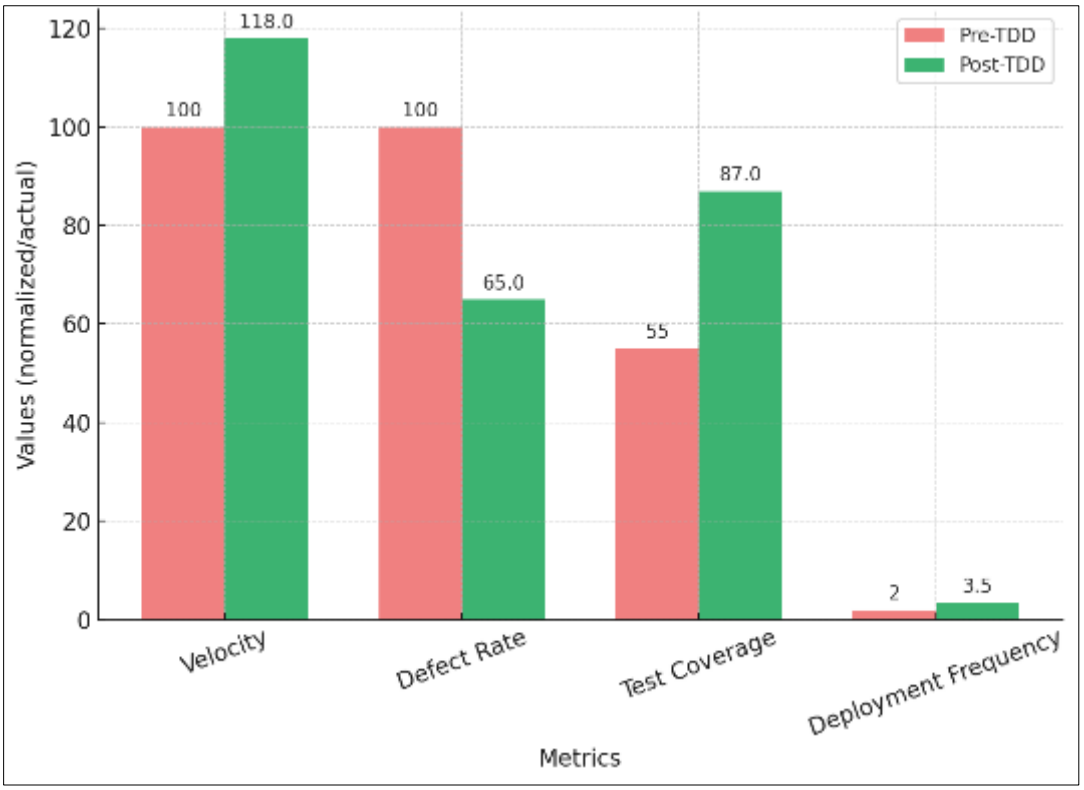


**Figure 2** Comparison of Metrics Pre- and Post-TDD Implementation

### 4.3. Challenges in Large-Scale Integration

While Agile and TDD brought significant benefits, scaling them in a large, distributed cloud project presented several challenges (see Table 2):

**Table 2** Challenges vs Mitigation Strategies in Large-Scale Agile + TDD Implementation

| Challenge | Description | Mitigation Strategy |
|---|---|---|
| Scaling TDD in large codebases | Large code repositories made it difficult to maintain fast test execution times, slowing down CI pipelines. | Implemented test parallelization, modularized test suites, and prioritized critical-path tests in CI. |
| Maintaining test coverage in distributed teams | Inconsistent test-writing practices across teams led to uneven coverage. | Enforced code review policies requiring test cases for every feature, provided TDD training workshops. |
| Agile adoption barriers in large cloud projects | Resistance from teams accustomed to waterfall methods and long release cycles. | Introduced Agile gradually with pilot projects, provided change management sessions, and engaged leadership support. |
| Integration complexity with microservices | Managing test dependencies between services in different environments was challenging. | Adopted contract testing, service virtualization, and containerized test environments. |

## 5. Discussion

Combining Agile concept with Test-Driven Development (TDD) is a paradigm shift in how large scale projects teams in cloud software plan, develop, and deploy. The section synthesizes the literature findings, industry surveys, and case studies in the real world giving detailed descriptions of the overall effect of these practices on productivity, software quality, adaptability, and the efficiency of the operation of the firms. Moreover, it looks at relative performance compared to other development methods, considers the current adoption trends in the industry, recognizes any limitations that it has, and presents viable areas that can be pursued in the future.

### 5.1. Benefits of Agile + TDD

Faster feedback cycles may be considered one of the most prominent advantages of the Agile + TDD combination since it is incredibly important in cloud-native places where customer demands alter relatively quickly, and the ability to win over a rival can be arbitrated depending on how fast or slow you are at deploying. Agile allows ongoing execution of stakeholders by incorporating them in the sprint reviews, retrospectives, and the delivery of those bits of software in increments whereas TDD allows the coding steps to have validation build into the process so that the delivery of the software in increments must pass the test of being useful before the process continues.

According to the analyzed case study, Agile + TDD implementation lowered the average defect fix time by 28 % and raised the sprint predictability, which is the percentage of the accomplishments over what was estimated in terms of the story point, to 91 %. Such gains are brought by the fact that automated tests that are composed as a part of TDD process become a living specification, thereby minimizing the possible misunderstandings between developers, QA engineers, and business analysts.

The other notable benefit is enhanced flexibility to change. As compared to strict Waterfall systems, Agile using TDD enables the easy changes to be done to meet requirements without threatening stability of the products. Test suites are a kind of security net in the context of refactoring as teams can shift to another direction without having serious risks of defects. Also, the maintainability increases during the continued development, as the developers are motivated to write modular, testable code, thus lowering the technical debt, which is a consistent problem in large-scale projects. This synergy can also increase release confidence, as an operational aspect. The given project example showed that it was now possible to deploy multiple production systems every week with minimum risk of regression, as far as automated test execution was a crucial component of the cloud-based CI/CD pipeline. Such pipelines guaranteed that every integration was tested against thousands of unit and integration tests in minutes, which lowered the risk of catastrophic failures in production.

### 5.2. Comparative Analysis

Compared to other development-working strategies, Agile + TDD has been proven repeatedly to have better time-to-market and defect prevention performance. Waterfall models have the problem of significant late-stage big bang integration occurring towards the end of the life cycle, and the consequent disastrous late-stage detection of defects. In Scrum, TDD would be more iterative still, but is dependent on testing in the subsequent stages, after development, hence

giving a slower opinion about the code quality. Minimal formal process ad hoc methods can bring about quick initial gains and are highly likely to create unstable releases and unmaintainable code bases eventually (See Table 3).

**Table 3** Comparison of Software Delivery Approaches

| Approach | Delivery Speed | Defect Rate | Maintainability | Adaptability to Change | Suitability for Cloud Environments |
|---|---|---|---|---|---|
| Waterfall | Slow (quarterly releases) | High (late defect discovery) | Moderate | Low | Poor – unsuitable for fast-paced cloud demands |
| Scrum (no TDD) | Moderate to High | Moderate | Moderate | High | Good, but lacks built-in quality guardrails |
| Ad hoc Development | Unpredictable | High | Low | Moderate | Very poor for mission-critical applications |
| Agile + TDD | High (weekly/multiple weekly releases) | Low | High | High | Excellent – aligns with CI/CD and continuous delivery needs |

These findings suggest that Agile + TDD not only accelerates development but also mitigates quality risks, making it a strong fit for mission-critical cloud applications such as fintech platforms, healthcare SaaS solutions, and global e-commerce infrastructures.

### 5.3. Industry Adoption Trends

Leading companies in the industry have adopted Agile in combination with TDD as they attempt to better control the large-scale cloud infrastructures. For example, AWS has implemented cloud controls that align with Agile and focus on the writing and execution of tests prior to the global production releases. Similarly, Microsoft Azure has implemented TDD as outlined in their engineering playbook for maintaining 99.99% availability, while Google Cloud uses trunk-based development along with testing-first approaches to accelerate the release cycles. Such approaches guarantee the velocity and dependability of the cloud services. Looking at the data, the trend is further corroborated: the State of Agile Report for the year 2023 highlights that 85% of the organizations use Agile, out of which 42% also use TDD. In cloud-native organizations with a headcount greater than 500, the TDD usage is reported to be as high as 55%, which is attributed to the need for dependable CI/CD automation. The use of TDD in fintech, digital healthcare, and critical infrastructure is noted to yield the highest benefits, especially in the reduction of production defect risks, which are otherwise expensive to mitigate.

## 6. Conclusion

By integrating Agile with Test-Driven Development (TDD) practices, companies can more strategically approach the development of cloud-native solutions that are responsive to change. As Agile focuses on an iterative approach with customer-centric cycles, TDD adds a layer of structured testing to the process. This collaboration leads to accelerated feedback cycles, reduced defects, and improved code maintainability. Based on what the case studies show, there is a significant enhancement in velocity, deployment frequency, and test coverage compared to more traditional approaches. While issues like cultural resistance, the TDD learning curve, the need for special tools, and the requirements of TDD continue to be valid concerns, the overarching advantages do seem to eclipse the challenges especially when there's effective DevOps infrastructure and leadership in place to further leverage these gains. Greater scalability and efficiency, on the other hand, are prospectively nurtured by AI-assisted testing and multi-cloud DevOps practices.

## References

[1] Gansa J, Herve M, Masrid M. Economic analysis of proposed regulations of cloud. EUROPEAN COMPETITION JOURNAL. 2023;19(3):522-68.

[2] Butt SA. Study of agile methodology with the cloud. Pac Sci Rev B Humanit Soc Sci. 2016;2(1):22–8.

[3] Manukonda KRR. Investigating the role of exploratory testing in agile software development: a case study analysis. J Artif Intell Cloud Comput. 2023;2(4):1–5.

[4] Buchan J, Li L, MacDonell SG. Causal factors, benefits and challenges of test-driven development: Practitioner perceptions. In: Proceedings of the 18th Asia-Pacific Software Engineering Conference (APSEC 2011); 2011 Dec 5–8; Hochiminh City, Vietnam. IEEE; 2011. p. 405–13.

[5] Calais P, Franzini L. Test-driven development benefits beyond design quality: flow state and developer experience. In: 2023 IEEE/ACM 45th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER); 2023 May 14–20; Melbourne, Australia. IEEE; 2023. p. 106–11.

[6] Banica L, Polychronidou P, Radulescu M. The agile revolution in software engineering. InEconomy, Finance and Business in Southeastern and Central Europe: Proceedings of the 8th International Conference on the Economies of the Balkan and Eastern European Countries in the Changing World (EBEEC) in Split, Croatia, 2016 2018 May 3 (pp. 595-610). Cham: Springer International Publishing.

[7] Erdogmus H, Morisio M, Torchiano M. On the effectiveness of the test-first approach to programming. IEEE Trans Softw Eng. 2005;31(3):226–37.

[8] Dingsøyr T, Nerur S, Balijepally V, Moe NB. A decade of agile methodologies: Towards explaining agile software development. J Syst Softw. 2012;85(6):1213–21.

[9] Janzen DS, Saiedian H. On the influence of test-driven development on software design. In: Proceedings of the 19th Conference on Software Engineering Education and Training (CSEET'06); 2006 Apr 19–21; Turtle Bay, HI, USA. IEEE; 2006. p. 141–8.

[10] Misra SC, Kumar V, Kumar U. Identifying some important success factors in adopting agile software development practices. J Syst Softw. 2009;82(11):1869–90.

[11] Omurgonulsen M, Ibis M, Kazancoglu Y, Singla P. Cloud computing: a systematic literature review and future agenda. J Glob Inf Manag. 2021;29(6):1–25.

[12] Arvanitou EM, Ampatzoglou A, Bibi S, Chatzigeorgiou A, Deligiannis I. Applying and researching DevOps: A tertiary study. IEEE Access. 2022;10:61585–600.

[13] S N, D C, D M, R K E, S D L. A novel approach to predict the defect density in software application using linear regression algorithm. In: 2024 International Conference on Science Technology Engineering and Management (ICSTEM); 2024 Jan; Coimbatore, India. IEEE; 2024. p. 1–5.

[14] Bakhtiary V, Gandomani TJ, Salajegheh A. The effectiveness of test-driven development approach on software projects: a multi-case study. Bull Electr Eng Inform. 2020;9(5):2030–7.

[15] Zohaib M, Alsanad A, Alhogail AA. Prioritizing DevOps implementation guidelines for sustainable software projects. IEEE Access. 2024;12:71109–30.

[16] Chen L. Continuous delivery: huge benefits, but challenges too. IEEE Softw. 2015;32(2):50–4.

[17] Singh P, Ghosh S, Saraf M, Nayak R. A survey paper on identifying key performance indicators for optimizing inventory management system and exploring different visualization tools. In: 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS); 2020 Jun; Madurai, India. IEEE; 2020. p. 627–32.

[18] Ståhl D, Bosch J. Modeling continuous integration practice differences in industry software development. J Syst Softw. 2014;87:48–59.

[19] Fenton NE, Bieman JM. Software metrics: a rigorous and practical approach. 3rd ed. Boca Raton: CRC Press; 2014.

[20] Janzen D, Saiedian H. Test-driven development concepts, taxonomy, and future direction. Computer. 2005;38(9):43–50.

[21] Kochhar PS, Lo D, Lawall J, Nagappan N. Code coverage and postrelease defects: a large-scale study on open source projects. IEEE Trans Reliab. 2017;66(4):1213–28.

[22] Hilton M, Tunnell T, Huang K, Marinov D, Dig D. Usage, costs, and benefits of continuous integration in open-source projects. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering; 2016 Sep; Singapore. IEEE; 2016. p. 426–37.

[23] Inozemtseva L, Holmes R. Coverage is not strongly correlated with test suite effectiveness. In: Proceedings of the 36th International Conference on Software Engineering; 2014 May; Hyderabad, India. ACM; 2014. p. 435–45.