ETHIOPIAN INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY RESEARCH

pISSN:2349-5707 Volume:13, Issue 01, January-2026 eISSN:2349-5715

Generative Intelligence In Behavior Driven Development: A Theoretical And Empirical Reframing Of Agile Test Automation In Contemporary Software Engineering

Arjun Malhotra
University of Melbourne, Australia

ABSTRACT: The increasing complexity of modern software systems has intensified the demand for development and testing methodologies that ensure both speed and quality. Behavior Driven Development has emerged as one of the most influential paradigms in this regard, aligning technical development with business-level expectations through structured behavioral specifications and executable acceptance tests. However, despite its conceptual appeal and practical successes, Behavior Driven Development faces persistent challenges related to specification maintenance, test scalability, duplication, natural language ambiguity, and the cognitive load placed on cross functional teams. In recent years, generative artificial intelligence has begun to reshape the broader software engineering landscape, offering unprecedented capabilities in natural language understanding, automated code generation, and adaptive learning from large scale datasets. Within this evolving context, the integration of generative intelligence into Behavior Driven Development represents not merely a tool level enhancement but a paradigm level transformation of how requirements, tests, and executable specifications are conceived, authored, maintained, and evolved. This research article develops a comprehensive theoretical and methodological analysis of how generative artificial intelligence can be systematically integrated into Behavior Driven Development to enhance test automation, stakeholder alignment, and long term maintainability. Grounded in the recent conceptual framework proposed by Tiwari (2025), this study situates generative automation of Behavior Driven Development within the broader historical evolution of agile, test driven, and behavior driven methodologies. Drawing upon empirical insights from prior industrial and academic studies, including large scale agile adoption research, systematic literature reviews, and domain specific BDD implementations, the article constructs an interpretive synthesis of the mechanisms through which generative models can improve requirement elicitation, scenario authoring, test coverage, and defect detection.

Rather than presenting numerical experiments, the study adopts a rigorous qualitative and conceptual methodology, integrating theoretical constructs from domain driven design, agile governance, and software quality assurance. The results demonstrate that generative automation significantly reduces ambiguity in behavioral specifications, mitigates test case duplication, supports continuous refactoring of BDD artifacts, and enables adaptive evolution of acceptance tests as systems change. At the same time, the research identifies critical limitations, including risks of over automation, semantic drift, and dependence on training data quality, thereby underscoring the need for human centered governance of generative systems. The article concludes by outlining a forward looking research agenda that positions generative BDD not as a replacement for human judgment, but as an augmentative intelligence layer that strengthens the epistemic foundations of agile software development.

Keywords
Behavior Driven Development, Generative Artificial Intelligence, Agile Testing, Automated Acceptance Testing, Software Quality Engineering, Test Automation, Domain Driven Design

INTRODUCTION

Software engineering has always been a discipline shaped by the tension between human intention and machine execution. From the earliest procedural programming paradigms to contemporary cloud native architectures, the central challenge has remained the same: how to ensure that what is built by machines truly reflects what is needed by people. Agile software development emerged in the late twentieth and early twenty first centuries as a response to this challenge, emphasizing adaptability, stakeholder collaboration, and iterative value delivery over rigid documentation and linear planning (Vijayasarathy and Turk, 2008; Dybå and Dingsøyr, 2008). Yet even within agile frameworks, the problem of translating human requirements into executable systems has persisted, often manifesting in the form of misinterpreted user stories, brittle tests, and a growing disconnect between business language and technical artifacts (Senapathi and Srinivasan, 2014; Alaidaros et al., 2019).

Behavior Driven Development emerged as a powerful methodological response to this disconnect. Rooted in the principles of Test Driven Development but oriented more explicitly toward business behavior, BDD reframes requirements as executable specifications written in a structured natural language format that can be understood by both technical and non technical stakeholders (Smart, 2014; Evans, 2004). By organizing system behavior into scenarios that follow the Given When Then structure, BDD seeks to create a shared language that bridges the cognitive gap between domain experts, developers, and testers. This shared language, when properly implemented, becomes a living documentation of system intent, continuously validated through automated acceptance tests (Rahman and Gao, 2015; Lopez Pellicer et al., 2014).

Despite its promise, the practical adoption of BDD has been far from trivial. Empirical studies have shown that while BDD can improve communication and early defect detection, it also introduces new complexities related to scenario maintenance, duplication, and scaling in large projects (Binamungu et al., 2018; Irshad et al., 2021; Pereira et al., 2018). As BDD specifications grow, teams struggle to keep them consistent, free of redundancy, and aligned with evolving system behavior. The cognitive effort required to write and maintain high quality scenarios can become a bottleneck, particularly in environments characterized by frequent change and rapid delivery cycles (Couto et al., 2020; Egbreghts, 2017).

It is within this context that generative artificial intelligence has begun to exert a transformative influence. Large scale generative models trained on vast corpora of natural language and code have demonstrated the ability to understand context, generate coherent text, and produce executable artifacts that align with human intent. In the domain of software engineering, these models have already been applied to code generation, bug detection, and documentation synthesis. However, their potential impact on Behavior Driven Development is only beginning to be systematically theorized. Tiwari (2025) provides one of the first comprehensive frameworks for automating BDD through generative AI, arguing that such systems can significantly enhance efficiency, reduce human error, and improve the alignment between requirements and tests.

The importance of this contribution cannot be overstated. By enabling the automated generation, validation, and evolution of BDD scenarios, generative AI has the potential to resolve many of the longstanding challenges that have constrained the scalability and sustainability of BDD in industrial settings. At the same time, the introduction of generative intelligence raises new epistemic, organizational, and ethical questions. How can teams ensure that AI generated scenarios truly reflect stakeholder intent? How should responsibility and accountability be distributed when tests are authored by machines? And how can the interpretability of generative systems be preserved within the collaborative ethos of agile development?

This article seeks to address these questions by developing a comprehensive research narrative that integrates Tiwari's (2025) generative BDD framework with the broader body of literature on agile development, automated testing, and behavior driven methodologies. Rather than treating generative AI as a mere technical add on, the study conceptualizes it as a socio technical innovation that reshapes the very foundations of requirement engineering and test automation. Through an extensive theoretical elaboration and critical synthesis of existing research, the article identifies the core mechanisms through which generative intelligence enhances BDD and delineates the conditions under which these enhancements can be realized without undermining the human centered values of agile practice.

The literature on agile adoption provides a crucial backdrop for this analysis. Early studies of agile teams highlighted the importance of cultural readiness, stakeholder engagement, and organizational support in determining the success of agile methods (Vijayasarathy and Turk, 2008; Murugaiyan and Balaji, 2012). Later empirical investigations confirmed that agile practices are most effective when they are embedded within a learning oriented organizational culture that values continuous improvement and cross functional collaboration (Senapathi and Srinivasan, 2014; Dybå and Dingsøyr, 2008). BDD, as an extension of agile thinking, inherits both the strengths and vulnerabilities of this cultural framework. When teams lack a shared understanding of domain concepts or when communication channels are fragmented, BDD scenarios can devolve into poorly structured, redundant, or misleading artifacts (Irshad et al., 2022; Binamungu et al., 2018).

Systematic literature reviews further reveal that the benefits of BDD, including improved requirement clarity and higher test coverage, are often counterbalanced by challenges related to tool integration, scenario evolution, and duplication detection (Abushama et al., 2021; Carrera et al., 2014; Romero Pena et al., 2021). These challenges are not merely technical but epistemological: they reflect the difficulty of maintaining a coherent shared understanding of system behavior as projects grow in size and complexity. Generative AI, by virtue of its capacity to model linguistic patterns and semantic relationships, offers a new way of addressing this epistemological burden.

Tiwari (2025) argues that generative models can be trained on domain specific corpora of BDD scenarios, user stories, and historical test outcomes, enabling them to produce contextually relevant and semantically consistent specifications. This capability transforms BDD from a labor intensive manual practice into a semi autonomous, continuously learning system of behavioral modeling. By automating routine aspects of scenario creation and maintenance, generative AI frees human practitioners to focus on higher level conceptual and ethical decisions, thereby strengthening rather than weakening the collaborative ethos of agile development.

The present study builds upon this insight by situating generative BDD within the broader theoretical frameworks of domain driven design and software quality engineering. Domain driven design emphasizes the importance of a ubiquitous language that reflects the core concepts of the business domain and is shared across all stakeholders (Evans, 2004). BDD operationalizes this principle by embedding the ubiquitous language into executable scenarios. Generative AI extends this operationalization by dynamically learning and refining the ubiquitous language based on ongoing project artifacts and feedback, thereby creating a living, adaptive representation of domain knowledge.

In light of these considerations, the central research problem addressed in this article can be formulated as follows: how can generative artificial intelligence be systematically integrated into Behavior Driven Development to enhance test automation, maintain semantic coherence, and support the long term evolution of agile software systems? By addressing this problem through an extensive synthesis of theory and empirical evidence, the study aims to contribute a foundational perspective that can guide both researchers and practitioners in navigating the emerging landscape of generative agile testing.

The remainder of the article unfolds as an integrated narrative. The methodology section explains the qualitative and interpretive research design used to synthesize existing literature and conceptual frameworks. The results section articulates the emergent patterns and themes that define generative BDD as a new paradigm of software testing. The discussion section engages in a deep theoretical analysis of these patterns, comparing competing scholarly viewpoints and exploring their implications for future research and practice. The conclusion distills the core insights of the study and outlines a forward looking agenda for the continued evolution of Behavior Driven Development in the age of generative intelligence.

METHODOLOGY

The methodological foundation of this study is rooted in interpretive software engineering research, drawing on principles of systematic literature synthesis, conceptual analysis, and theory building as articulated in established guidelines for software engineering reviews (Keele, 2007; Petersen et al., 2008). Rather than relying on experimental or quantitative data, the present research seeks to develop a rich, theoretically grounded understanding of how generative artificial intelligence transforms Behavior Driven Development by integrating insights from diverse strands of existing scholarship. This approach is particularly appropriate given the emergent nature of generative BDD, which has not yet reached a level of maturity that would permit large scale controlled experimentation across multiple industrial contexts.

The first methodological pillar of the study involves a comprehensive interpretive synthesis of the literature on Behavior Driven Development, agile testing, and automated acceptance testing. Foundational works on agile software development provide the historical and conceptual baseline against which BDD is understood (Schwaber, 2004; Vijayasarathy and Turk, 2008; Murugaiyan and Balaji, 2012). Empirical studies and systematic reviews then offer insights into how BDD has been adopted, adapted, and challenged in practice (Dybå and Dingsøyr, 2008; Abushama et al., 2021; Alaidaros et al., 2019). Within this corpus, particular attention is given to studies that analyze the maintenance, refactoring, and duplication of BDD specifications, as these represent the key pain points that generative AI seeks to address (Binamungu et al., 2018; Borg and Kropp, 2011; Irshad et al., 2022; Xu et al., 2021).

The second methodological pillar involves the integration of generative AI theory into the BDD discourse. Tiwari (2025) serves as the conceptual anchor for this integration, providing a structured framework for understanding how generative models can automate and enhance BDD workflows. Rather than treating this reference as an isolated contribution, the study situates it within a broader landscape of natural language processing and assisted BDD research, including earlier attempts to use linguistic analysis to support scenario generation and validation (Soeken et al., 2012; Smart, 2014). By comparing these earlier approaches with the capabilities of modern generative models, the methodology identifies both continuities and discontinuities in the evolution of automated BDD.

A third methodological component is conceptual triangulation. This involves comparing insights from BDD research with parallel developments in requirements engineering, learning analytics, and educational technology, which offer analogous challenges related to the interpretation and automation of human generated artifacts (Ouhbi et al., 2015; Omer et al., 2023; Farooq et al., 2022). While these domains are not identical to software testing, they provide valuable theoretical lenses through which the dynamics of generative specification authoring can be better understood.

The analytic process itself proceeds through iterative thematic coding and theory building. Core themes such as semantic coherence, duplication management, stakeholder alignment, and test evolution are identified across the literature and examined in relation to generative AI capabilities as articulated by Tiwari (2025). These themes are then woven into a coherent explanatory framework that accounts for both the opportunities and the limitations of generative BDD. Throughout this process, scholarly debate and counter arguments are explicitly incorporated, ensuring that the resulting theory does not merely celebrate technological innovation but critically engages with its potential risks and trade offs (Pereira et al., 2018;

Egbreghts, 2017).

The methodological rigor of the study is further supported by adherence to established principles of systematic mapping and review in software engineering. Although the present work is not a formal systematic literature review, it draws on the methodological insights of Petersen et al. (2008) and Keele (2007) to ensure transparency, comprehensiveness, and analytical depth. By integrating diverse sources in a structured and theoretically informed manner, the study aims to produce findings that are both credible and generative of future research.

Like all interpretive research, this study has limitations. Its reliance on existing literature and conceptual analysis means that its conclusions are necessarily provisional and subject to revision as new empirical data emerges. Moreover, the rapid evolution of generative AI technologies means that specific technical capabilities may change faster than scholarly consensus can be established. Nevertheless, by grounding its analysis in well established theories of agile development and BDD, and by anchoring its generative AI perspective in the framework proposed by Tiwari (2025), the study provides a robust foundation for understanding the current and future role of generative intelligence in Behavior Driven Development.

RESULTS

The interpretive synthesis of the literature reveals a coherent set of patterns that collectively define generative Behavior Driven Development as a distinct and transformative paradigm within agile software engineering. These patterns emerge from the intersection of longstanding challenges in BDD practice and the novel affordances introduced by generative artificial intelligence, as articulated in both empirical studies and conceptual frameworks (Tiwari, 2025; Irshad et al., 2021; Binamungu et al., 2018).

One of the most prominent findings concerns the reduction of semantic ambiguity in behavioral specifications. Traditional BDD relies on human authored natural language scenarios, which are inherently prone to vagueness, inconsistency, and misinterpretation, particularly when multiple stakeholders contribute to a growing corpus of specifications (Smart, 2014; Pereira et al., 2018). Generative AI models, trained on large datasets of domain specific language and historical test outcomes, can infer latent semantic structures and enforce a higher degree of linguistic consistency across scenarios (Tiwari, 2025; Soeken et al., 2012). This capability does not eliminate the need for human judgment, but it provides a powerful assistive layer that flags potential ambiguities and suggests more precise formulations, thereby improving the epistemic quality of BDD artifacts.

A second major pattern involves the mitigation of test case duplication and redundancy. Empirical studies have consistently shown that as BDD suites grow, duplicate or near duplicate scenarios proliferate, increasing maintenance costs and obscuring the true behavioral coverage of the system (Binamungu et al., 2018; Romero Pena et al., 2021; Irshad et al., 2022). Generative AI systems can analyze semantic similarity across large numbers of scenarios, identifying overlaps that would be difficult for human reviewers to detect. Tiwari (2025) demonstrates that generative models can not only detect duplication but also propose refactored, generalized scenarios that preserve behavioral intent while reducing redundancy. This finding aligns with earlier work on automated acceptance test refactoring, but extends it by leveraging deep semantic modeling rather than rule based heuristics (Borg and Kropp, 2011).

A third result pertains to the dynamic evolution of BDD specifications in response to system change. In traditional practice, updating scenarios to reflect new or modified requirements is a labor intensive process that often lags behind actual code changes, leading to brittle or outdated tests (Rahman and Gao, 2015; Irshad et al., 2021). Generative AI introduces the possibility of continuous, adaptive scenario evolution. By monitoring code repositories, test outcomes, and user stories, a generative system can suggest updates to BDD scenarios that maintain alignment with the current state of the system (Tiwari, 2025; Xu et al., 2021). This capability transforms BDD from a static documentation practice into a living, co evolving model of system behavior.

Another significant pattern concerns stakeholder collaboration and accessibility. One of the original promises of BDD was to create a shared language that could be understood by business stakeholders as well as technical teams (Evans, 2004; Smart, 2014). In practice, however, the technical syntax of many BDD tools has limited this accessibility, reinforcing rather than dissolving the boundary between business and development (Pereira et al., 2018; Couto et al., 2020). Generative AI systems, by contrast, can translate between informal stakeholder narratives and formal Given When Then scenarios, effectively acting as linguistic mediators that enhance cross functional communication (Tiwari, 2025; Soeken et al., 2012). This mediating role strengthens the socio technical fabric of agile teams by enabling more inclusive participation in the specification process.

Finally, the results reveal a complex pattern of dependency and governance. While generative AI can significantly enhance the efficiency and quality of BDD, it also introduces new dependencies on training data, model transparency, and organizational oversight (Egbreghts, 2017; Abushama et al., 2021). Tiwari (2025) emphasizes that without proper human in the loop mechanisms, generative systems risk producing plausible but incorrect or misaligned scenarios. This finding underscores the importance of integrating generative BDD within a robust governance framework that preserves human accountability and domain expertise.

Collectively, these results suggest that generative BDD is not simply an incremental improvement to existing test automation practices but a qualitatively new mode of software specification and validation. By embedding deep linguistic and contextual intelligence into the heart of the BDD workflow, generative AI redefines what it means for software behavior to be specified, tested, and understood within agile organizations.

DISCUSSION

The emergence of generative Behavior Driven Development represents a profound shift in the epistemology of software engineering. At its core, BDD has always been about making behavior explicit, shared, and verifiable through executable narratives (Smart, 2014; Evans, 2004). What generative AI adds to this paradigm is not merely speed or convenience, but a new form of machine mediated understanding that operates at the level of meaning rather than syntax. This section explores the theoretical, practical, and ethical implications of this shift by engaging in a deep dialogue with the existing literature and the framework articulated by Tiwari (2025).

From a theoretical standpoint, generative BDD can be understood as the convergence of two historically distinct traditions in software engineering: the formalist tradition of automated testing and the interpretivist tradition of requirements engineering. Automated testing has long sought to minimize human intervention by encoding expected behavior in executable form, thereby enabling machines to verify correctness at scale (Tuteja and Dubey, 2012; Rahman and Gao, 2015). Requirements engineering, by contrast, has emphasized the inherently social and interpretive nature of understanding what a system should do, acknowledging that stakeholder needs are often ambiguous, evolving, and context dependent (Ouhbi et al., 2015; Dybå and Dingsøyr, 2008). BDD bridged these traditions by embedding natural language requirements within automated tests. Generative AI now deepens this bridge by enabling machines to participate in the interpretive act itself.

This participation, however, is not without controversy. Critics of AI assisted software engineering argue that delegating interpretive tasks to machines risks eroding the reflective practices that are central to agile development (Pereira et al., 2018; Egbreghts, 2017). If scenarios are generated automatically, will teams still engage deeply with the underlying business logic, or will they accept machine produced artifacts uncritically? Tiwari (2025) addresses this concern by emphasizing the role of generative AI as an augmentative rather than substitutive intelligence. In this view, generative systems propose, but humans dispose. The value of the technology lies in its ability to surface patterns, inconsistencies, and possibilities that might otherwise remain hidden, thereby enriching rather than impoverishing human understanding.

The discussion of duplication management provides a concrete illustration of this dynamic. Empirical studies have shown that duplicate BDD scenarios are not merely a technical nuisance but a symptom of deeper organizational and cognitive fragmentation (Binamungu et al., 2018; Romero Pena et al., 2021). Different teams may describe the same behavior in slightly different ways, reflecting divergent mental models of the system. A generative AI that detects and consolidates these duplicates does more than clean up the test suite; it reveals and potentially reconciles these divergent models. In doing so, it supports the creation of a more coherent shared understanding, which is the ultimate goal of BDD (Evans, 2004; Smart, 2014).

At the same time, the risk of semantic drift must be acknowledged. Generative models learn from historical data, and if that data reflects outdated or suboptimal practices, the model may perpetuate them (Tiwari, 2025; Abushama et al., 2021). This raises important questions about model governance, versioning, and validation. Just as code is subject to review and refactoring, so too must generative models and their training datasets be continuously evaluated to ensure alignment with current domain knowledge and organizational values.

Another critical dimension of the discussion concerns scalability. Large scale software systems pose unique challenges for BDD, including the coordination of thousands of scenarios across multiple teams and the integration of BDD with other testing and deployment pipelines (Irshad et al., 2021; Irshad et al., 2022). Generative AI offers powerful tools for managing this complexity by automating scenario generation, refactoring, and alignment. However, scalability is not merely a technical property; it is also organizational. The introduction of generative BDD requires new roles, skills, and processes to oversee and integrate AI outputs into the agile workflow. Without such organizational adaptation, the technology may fail to deliver its promised benefits.

The implications for education and professional development are equally significant. As learning analytics and blended learning models reshape how software engineers are trained (Omer et al., 2023; Farooq et al., 2022), generative BDD tools could become integral to teaching the principles of requirements specification and test driven thinking. By providing immediate, context aware feedback on scenario quality and coverage, generative systems can support a more experiential and reflective learning process. Yet this potential will only be realized if educators and practitioners remain critically engaged with the tools, rather than treating them as infallible oracles.

Ultimately, the debate around generative BDD reflects a broader tension within contemporary software engineering: the desire to harness the power of automation without sacrificing the human judgment, creativity, and ethical responsibility that define the profession. Tiwari (2025) provides a compelling vision of how this balance can be achieved, but realizing that vision will require ongoing research, dialogue, and experimentation across the global software engineering community.

CONCLUSION

This research has argued that the integration of generative artificial intelligence into Behavior Driven Development constitutes a paradigm shift in how software behavior is specified, tested, and understood. By synthesizing the extensive literature on agile development, BDD, and automated testing with the generative framework articulated by Tiwari (2025), the study has shown that generative BDD offers powerful solutions to longstanding challenges of ambiguity, duplication, and maintenance. At the same time, it has highlighted the need for robust human centered governance to ensure that these technologies enhance rather than undermine the collaborative and interpretive foundations of agile practice.

As software systems continue to grow in complexity and societal impact, the demand for reliable, transparent, and adaptive testing methodologies will only intensify. Generative Behavior Driven Development, properly implemented, has the potential to meet this demand by creating a living, intelligent fabric of executable knowledge that evolves alongside the systems it describes. The future of agile testing, therefore, lies not in choosing between human and machine intelligence, but in designing socio technical systems that allow each to complement and strengthen the other.

REFERENCES

1. Senapathi, M., and Srinivasan, A. (2014). An empirical investigation of the factors affecting agile usage. Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, 1–10.

2. Romero Pena, I., Padilla Zarate, G., and Cortes Verdin, K. (2021). Identification of test cases duplication: Systematic literature review. Proceedings of the 9th International Conference in Software Engineering Research and Innovation, 104–111.

3. Evans, E. (2004). Domain driven design: Tackling complexity in the heart of software. Addison Wesley Professional.

4. Tiwari, S. K. (2025). Automating Behavior Driven Development with Generative AI: Enhancing efficiency in test automation. Frontiers in Emerging Computer Science and Information Technology, 2(12), 01–14.

5. Vijayasarathy, L., and Turk, D. (2008). Agile software development: A survey of early adopters. Journal of Information Technology Management, 19(2), 1–8.

6. Binamungu, L. P., Embury, S. M., and Konstantinou, N. (2018). Maintaining behaviour driven development specifications: Challenges and opportunities. IEEE International Conference on Software Analysis, Evolution and Reengineering, 175–184.

7. Irshad, M., Britto, R., and Petersen, K. (2021). Adapting Behavior Driven Development for large scale software systems. Journal of Systems and Software, 177, 110944.

8. Abushama, H. M., Alassam, H. A., and Elhaj, F. A. (2021). The effect of test driven development and behavior driven development on project success factors. International Conference on Computer, Control, Electrical, and Electronics Engineering, 1–9.

9. Smart, J. F. (2014). BDD in action: Behavior driven development for the whole software lifecycle. Manning Publications.

10. Dybå, T., and Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic

review. Information and Software Technology, 50(9–10), 833–859.

11. Keele, S. (2007). Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report.

12. Soeken, M., Wille, R., and Drechsler, R. (2012). Assisted behavior driven development using natural language processing. Objects, Models, Components, Patterns, 269–287.

13. Rahman, M., and Gao, J. (2015). A reusable automated acceptance testing architecture for microservices in behavior driven development. IEEE Symposium on Service Oriented System Engineering, 321–325.

14. Irshad, M., Borstler, J., and Petersen, K. (2022). Supporting refactoring of BDD specifications. Information and Software Technology, 141, 106717.

15. Pereira, L., Sharp, H., de Souza, C., Oliveira, G., Marczak, S., and Bastos, R. (2018). Behavior driven development benefits and challenges. International Conference on Agile Software Development Companion, 1–4.

16. Borg, R., and Kropp, M. (2011). Automated acceptance test refactoring. Workshop on Refactoring Tools, 15–21.

17. Xu, J., Du, Q., and Li, X. (2021). Requirement based regression test selection in BDD. IEEE Computers, Software, and Applications Conference, 1303–1308.

18. Couto, T., Marczak, S., and Gomes, F. (2020). Measuring the benefits of Behavior Driven Development adoption. Brazilian Symposium on Software Quality, 1–7.

19. Alaidaros, H., Omar, M., and Romli, R. (2019). Key factors of evaluating agile approaches. International Journal of Supply Chain Management, 8(2), 1–11.

20. Schwaber, K. (2004). Agile project management with Scrum. Microsoft Press.

21. Murugaiyan, M. S., and Balaji, S. (2012). Succeeding with agile software development. IEEE International Conference on Advances in Engineering, Science and Management, 162–165.

22. Ouhbi, S., Idri, A., Fernandez Aleman, J. L., and Toval, A. (2015). Requirements engineering education. Requirements Engineering, 20, 119–138.

23. Omer, U., Tehseen, R., Farooq, M. S., and Abid, A. (2023). Learning analytics in programming courses. Education and Information Technologies, 1–48.

24. Farooq, M. S., Hamid, A., Alvi, A., and Omer, U. (2022). Blended learning models and gamification in project management education. IEEE Access.

25. Egbreghts, A. (2017). A literature review of behavior driven development using grounded theory. Twente Student Conference on IT.