ISSN-3139-6593 (Online)

Journal of Interdisciplinary Science &Technology

Impact of Generative AI on Agile Software and Product Teams: A Structured Review and Human-in-the-Loop Adoption Framework

Ekansh Tayade[1*], Pranav Pagare[2], Darshan Aher[3], Smit Chaudhari[4]

[1,2,3] Computer Engineering, Sandip Institute of Technology and Research Center Nashik (SITRC), Maharashtra, India;

[4] Dharmsinh Desai University, Nadiad, Gujarat (DDU), India

ekanshshweta06@gmail.com, pranavnpagare@gmail.com, aherdarshan2803@gmail.com, smitchaudhari2601@gmail.com

Corresponding author- Ekansh Tayade

***DOI**: 10.5281/zenodo.22861446*

Abstract: Generative artificial intelligence (GenAI) is increasingly embedded in software engineering and product-development workflows, particularly through large language models that can generate code, tests, documentation, summaries, design alternatives, and natural-language explanations. This paper examines how such capabilities interact with Agile ways of working, where value is delivered through short feedback cycles, cross-functional collaboration, empirical learning, and shared ownership. A structured review of research and practitioner evidence is used to organize GenAI's effects across product discovery, requirements, backlog refinement, sprint planning, implementation, testing, review, release, and continuous improvement. Evidence indicates that AI assistance can accelerate selected development tasks and reduce friction in activities such as code generation and comprehension, while practitioner surveys also report perceived benefits in code quality, learning, and customer alignment. At the same time, faster generation can shift bottlenecks toward validation, integration, security review, architectural consistency, and decision quality. The paper therefore treats productivity as a multidimensional construct rather than a simple measure of output volume. A Human-in-the-Loop GenAI Agile Adoption Framework is proposed around five recurring controls: framing, generation, verification, decision, and learning. The framework connects product, engineering, quality, security, and governance responsibilities and introduces measurable adoption dimensions spanning delivery flow, quality, developer experience, product outcomes, and risk. The resulting model positions GenAI as an augmentation layer within Agile systems rather than as a substitute for human accountability.

Keywords: Generative AI, Large Language Models, Agile Software Development, Product Management, Developer Productivity, Human-in-the-Loop, AI Governance, Scrum, Software Engineering

I INTRODUCTION

Agile software and product teams are designed around rapid learning: work is decomposed into increments, feedback is gathered frequently, and decisions are revised as evidence changes. GenAI introduces a new form of assistance into this environment. Instead of automating one fixed step, a language model can participate across requirements analysis, coding, testing, documentation, planning, and communication. This breadth makes GenAI different from a conventional development utility: the same system may act as a drafting assistant, coding partner, reviewer, explainer, or brainstorming tool depending on context [3,10].

The central question is therefore not simply whether GenAI makes developers faster. Developer productivity research argues that productivity is multidimensional and includes individual activity, flow, collaboration, communication,

satisfaction, and organizational outcomes [2]. Agile teams add further dimensions: product value, feedback quality, sustainable pace, technical quality, and the ability to adapt. A tool can reduce typing time while increasing review effort or architectural inconsistency; conversely, it can reduce low-value work and create more time for design, customer learning, and collaboration.

This paper develops a structured synthesis of the impact of GenAI on Agile software and product teams. It focuses on

three questions: (1) where GenAI changes work across the Agile lifecycle; (2) which productivity and collaboration effects are supported by current evidence; and (3) what operating controls allow teams to capture benefits without weakening verification, security, accountability, or product judgment.

Research on AI pair programming suggests that developers do not use generated suggestions uniformly. Barke et al. describe distinct interaction modes in which developers may use an assistant to accelerate a well-understood task or to explore a solution space when requirements are less certain [6].

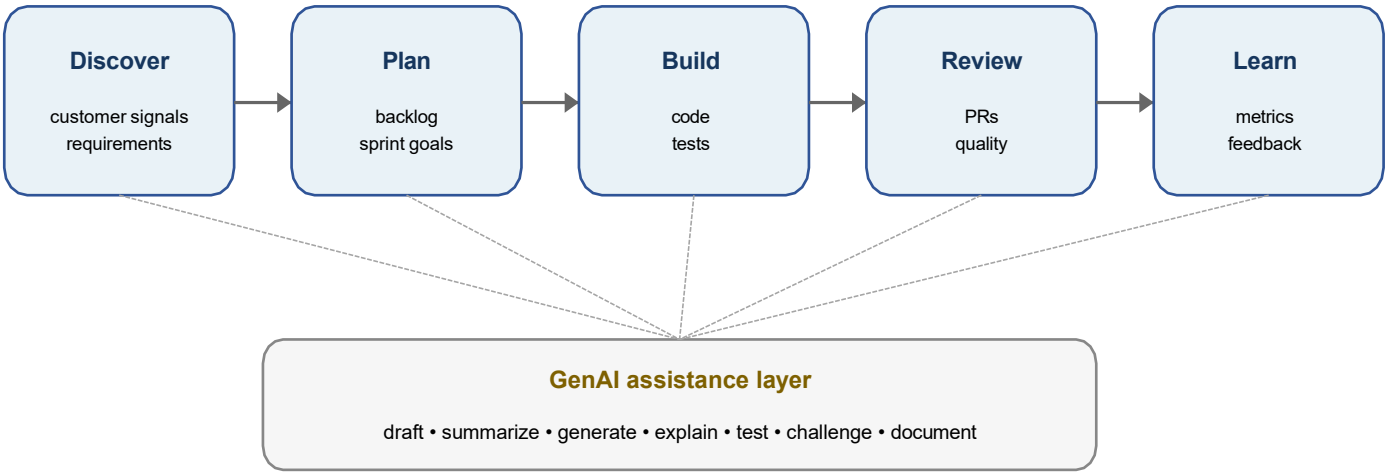

Fig. 1. GenAI assistance across the Agile product-development loop. The AI layer supports multiple activities, while humans retain intent, acceptance, and accountability.

II BACKGROUND AND RELATED WORK

A. Generative AI in Software Engineering

Large language models have demonstrated strong capabilities in code generation, code explanation, transformation, summarization, and natural-language interaction with software artifacts [3,4,15]. The software-engineering literature has expanded rapidly: Hou et al. reviewed 395 papers on large language models for software engineering and mapped applications across the development lifecycle [10]. This breadth matters for Agile teams because the value of GenAI is distributed across many small interactions rather than a single automated process as shown in Figure 1.

B. Agile Software and Product Teams

The Scrum Guide defines Scrum around empiricism, transparency, inspection, and adaptation, with accountabilities distributed across Product Owner, Scrum Master, and Developers [1]. In practice, Agile product teams also include design, data, security, quality, and domain specialists. GenAI interacts with this system by reducing the cost of producing intermediate artifacts. The resulting organizational challenge is to ensure that lower-cost generation does not weaken inspection and adaptation.

C. Human–AI Interaction

This distinction is important for Agile work: acceleration can be beneficial when acceptance criteria are stable, while exploration requires stronger human reasoning because the generated options may influence problem framing itself

III METHODOLOGY

The paper uses a structured literature-review approach. Sources were selected to cover five evidence groups: foundational Agile practice; software-engineering and code-generation research; developer-productivity studies; requirements and testing research; and governance or practitioner evidence. The synthesis does not treat all sources as equivalent. Controlled experiments are used for task-level causal evidence, systematic reviews for breadth, surveys for perceptions and adoption patterns, and governance frameworks for control requirements. Findings are organized thematically across the Agile lifecycle and then translated into an operating framework.

The resulting framework is deliberately evidence-aware. Where studies report a measured task effect, the effect is identified as such. Where surveys report perceptions, the paper describes them as perceptions. Where the framework proposes controls or metrics, those elements are presented as design recommendations derived from the synthesis rather than as experimentally validated outcomes.

IV EVIDENCE ON PRODUCTIVITY AND ADOPTION

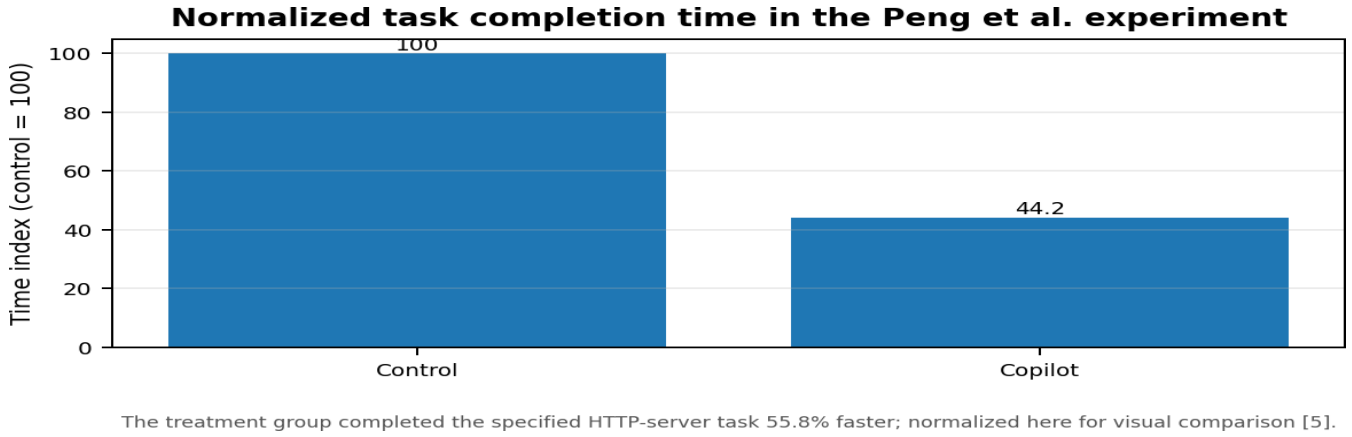

Fig. 2. Controlled task-level evidence from Peng et al. [5]. The normalized visualization preserves the reported 55.8% faster completion result without treating it as a general team-productivity estimate.

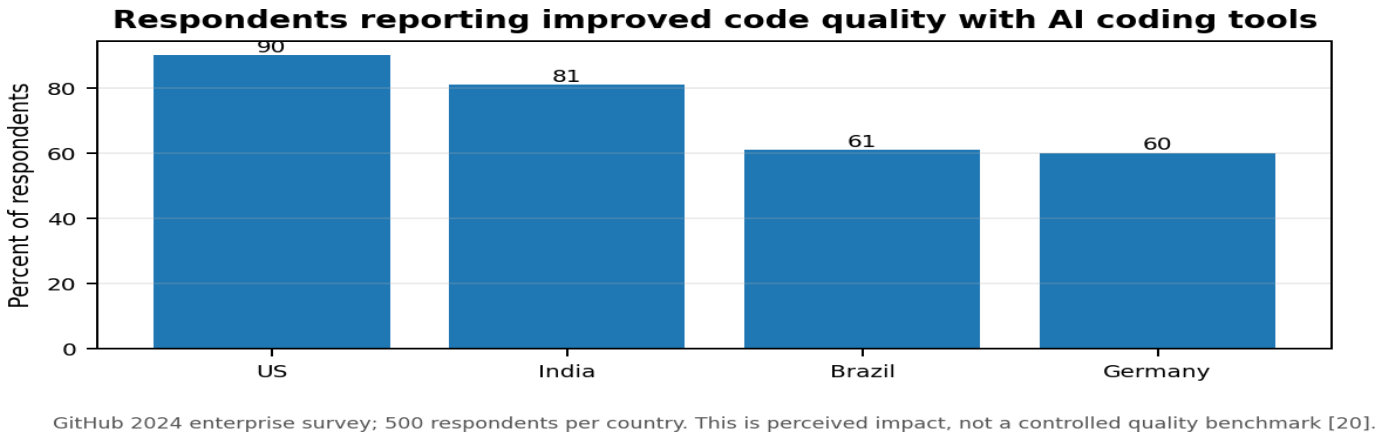

Fig. 3. Perceived code-quality improvement reported by GitHub's 2024 survey across four markets [20]. The survey population was enterprise respondents; these are perceptions rather than randomized quality measurements.

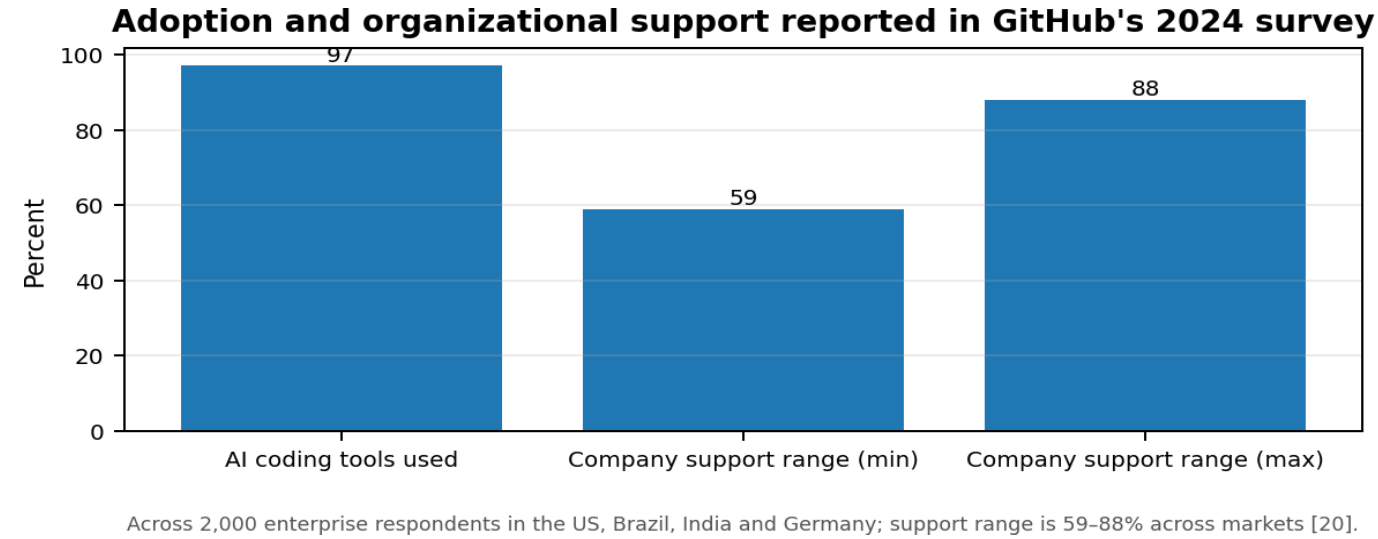

Fig. 4. AI coding-tool use and company support reported in GitHub's 2024 survey [20]. The 97% figure refers to respondents who had used AI coding tools; organizational support varied by market.

V GENAI ACROSS THE AGILE LIFECYCLE

A. Product Discovery and Requirements

GenAI can synthesize interview notes, cluster customer feedback, rewrite requirements into clearer language, identify missing acceptance criteria, and generate alternative problem statements. Requirements-engineering research indicates that automation can support activities such as elicitation, analysis, specification, and traceability [12,18]. The key limitation is that generated requirements can sound precise while encoding incorrect assumptions. Product teams should therefore treat generated requirements as candidate artifacts that require domain validation as shown in Figure 2, 3, 4.

B. Sprint Planning and Backlog Refinement

In backlog work, GenAI can summarize previous sprint outcomes, split large items into candidate stories, identify dependencies, propose acceptance criteria, and convert technical discussions into stakeholder-readable summaries. The principal benefit is reduction of preparation effort. The principal risk is false precision: a generated decomposition can create the appearance of completeness even when dependencies, non-functional requirements, or operational constraints are missing.

C. Development and Testing

Coding assistance is the most visible GenAI application. Controlled evidence from Peng et al. found that developers with Copilot completed a specified HTTP-server task 55.8% faster than the control group [5]. However, task-level acceleration should not be equated with end-to-end product velocity. Generated code can introduce defects, insecure patterns, dependency problems, or context mismatches; studies of AI pair programming therefore emphasize the importance of expertise and verification [7,9].

Testing is another high-leverage area. Models can draft unit tests, explain failing assertions, generate edge cases, and propose test data. Research has also explored language-model support for testing and metamorphic testing [13]. Yet generated tests can reproduce the implementation's assumptions and therefore fail to detect the very defect they should expose. Human review of test intent and independent validation remain necessary.

D. Review, Integration, and Release

As generation becomes cheaper, code review can become the limiting stage. Pull requests may contain more generated material, documentation may expand, and reviewers may face larger volumes of superficially plausible output. This shifts the objective from reviewing every line with equal intensity to applying risk-based verification. Security scanning, automated tests, dependency analysis, architecture checks, and targeted human review become complementary controls.

VI IMPACT ON PRODUCTIVITY AND COLLABORATION

A. Productivity Is Multi-Dimensional

The SPACE framework cautions against reducing developer productivity to a single output metric [2]. In GenAI-enabled teams, this becomes more important because code volume can increase without a corresponding increase in customer value. Useful measures include cycle time, review latency, change failure rate, defect escape rate, developer flow, cognitive load, collaboration, and customer outcomes. The appropriate interpretation depends on the team and product context.

B. Collaboration and Knowledge Sharing

GenAI can lower the cost of translating technical material into accessible language. A developer can ask for a codebase explanation; a product manager can request a plain-language summary of an implementation change; a new team member can use an assistant to navigate unfamiliar concepts. GitHub's 2023 developer survey reported that 81% of respondents expected AI coding tools to make teams more collaborative, while also emphasizing the importance of developer experience and collaboration [4]. Such results are expectations or perceptions, not proof that every team will collaborate better.

C. Pair Programming and Team Roles

AI pair programming changes the distribution of cognitive work. Developers may spend less time on syntax and boilerplate and more time on decomposition, review, debugging, and design. Product managers may use GenAI to prepare research summaries and communication artifacts. QA and security specialists can use it for test ideas and threat-model prompts. The team therefore becomes a system in which AI participates in multiple role-adjacent tasks, but professional accountability remains human.

VII IMPACT ON PRODUCT MANAGEMENT

A. Discovery and Customer Insight

Product teams can use GenAI to synthesize interview transcripts, support-ticket themes, survey responses, and market notes. The value comes from reducing synthesis effort rather than outsourcing product judgment. Generated clusters and summaries should be checked against source evidence because language models may merge distinct customer problems or amplify repeated but unrepresentative comments

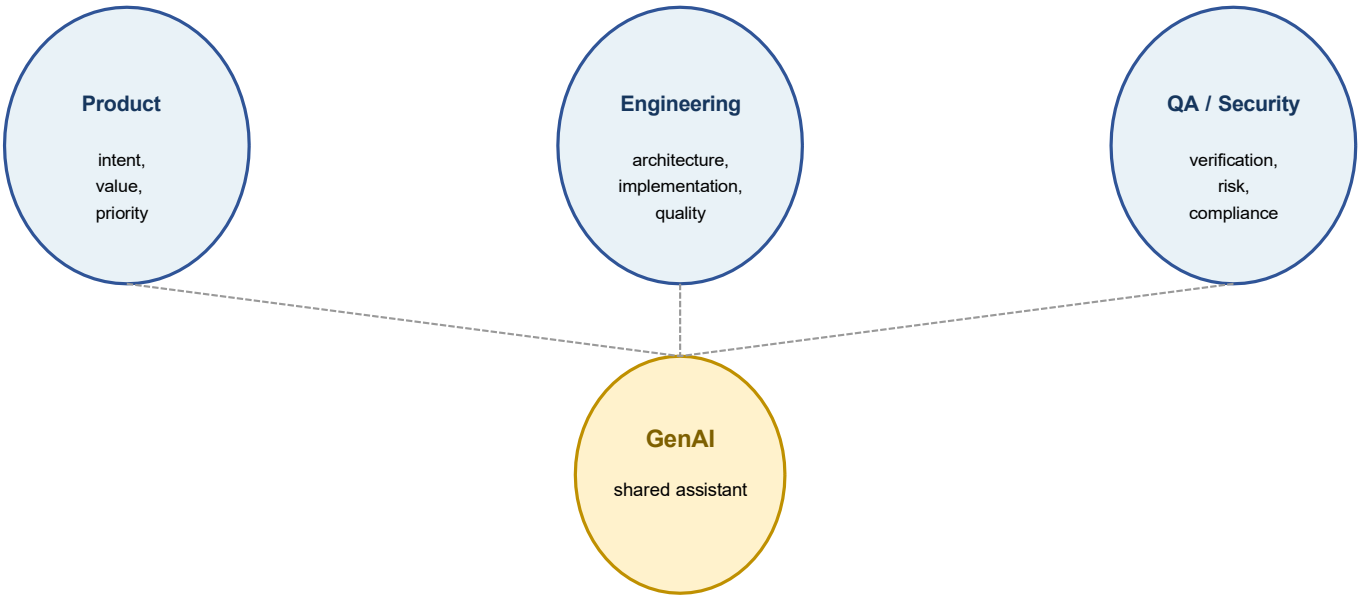

Fig. 5. Cross-functional GenAI support. The assistant can serve multiple roles simultaneously, but ownership of product intent, engineering decisions, and verification remains distributed across the team

B. Roadmapping and Prioritization

GenAI can transform strategic inputs into candidate roadmap themes, compare alternative sequencing scenarios, and expose assumptions. It should not independently determine priorities because prioritization involves business strategy, customer value, constraints, risk, and trade-offs that are not fully encoded in historical text. The recommended use is decision support: make assumptions explicit, generate alternatives, and require human selection.

C. Documentation and Communication

Documentation is particularly compatible with language-model assistance because much of the work involves transforming existing information into different formats. Release notes, architecture summaries, meeting notes, product briefs, and stakeholder updates can be drafted rapidly. Teams should maintain a source-of-truth policy so that generated documents do not become authoritative merely because they are well written.

D. Product Experiments

GenAI can accelerate experiment design by generating hypotheses, alternative copy, test cases, and analysis templates. However, faster experiment generation can also create experimental noise as depicted in Figure 5. Product teams should preserve explicit hypotheses, success criteria, sampling assumptions, and decision rules rather than treating generated variants as evidence of value.

VIII RISKS AND FAILURE MODES

A. Correctness and Hallucination

A language model can produce syntactically valid but semantically incorrect code, plausible explanations that are unsupported by source material, or invented references. Verification must therefore be designed around the artifact's risk: executable outputs require tests and static analysis; factual summaries require source checking; architectural proposals require review against constraints.

B. Security and Privacy

GenAI adoption introduces questions about what source code, customer information, credentials, logs, or proprietary documents are sent to external systems. NIST's AI Risk Management Framework provides a general structure for governing AI risks through Govern, Map, Measure, and Manage functions as depicted in Table 1 [14].

Risk- Control Matrix for Team Adoption

Risk	Typical Failure	Control
Hallucination	Plausible but wrong output	Tests + Source Checks
Security	Unsafe or leaked code/data	Approved tools + Scanning
Context Drift	Solution ignores Architecture	Repo context + Review
Over - Reliance	Acceptance without understanding	Explain+ reviewer Ownership
IP/License	Unclear Provenance	Policy+ dependency Review
Review Bottle Neck	More Output, Same Capacity	Risk Based review Queues

Table 1. Common GenAI risks and Corresponding Operating Controls

For software teams, this translates into approved tools, data-classification rules, access controls, logging, security testing, and incident response.

C. Over-Reliance and Skill Atrophy

If developers accept generated output without understanding it, short-term speed may weaken debugging and architectural reasoning. This is particularly relevant to early-career engineers. Teams can mitigate the risk through explanation requirements for high-risk changes, code-review ownership, deliberate learning tasks, and periodic work without AI assistance when appropriate.

D. Context and Architectural Drift

A model's response is only as good as the context supplied to it. Local coding suggestions can conflict with architectural conventions, domain rules, or operational constraints. Retrieval from approved repositories, explicit project instructions, architecture documentation, and reviewer checkpoints can reduce this problem but do not remove it.

E. Trust and Accountability

The central governance question is not who typed the code, but who accepted the change. Teams should preserve a clear human owner for requirements, architecture, security, and release decisions. AI-generated artifacts should be identifiable in the workflow where this is necessary for auditability and review as shown in figure 6.

IX HUMAN-IN-THE-LOOP GENAI AGILE ADOPTION FRAMEWORK

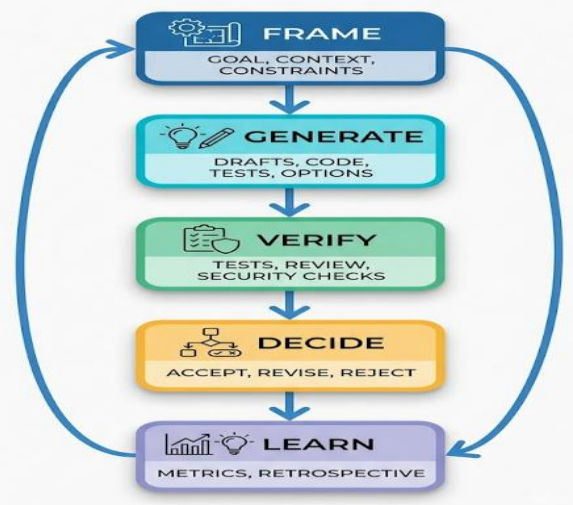

Fig. 6. Proposed Human-in-the-Loop operating loop

The proposed framework treats GenAI as a controlled assistance layer. **Frame** defines the problem, source context, constraints, and acceptance criteria. **Generate** produces candidate code, requirements, tests, summaries, or alternatives. **Verify** applies automated and human checks appropriate to risk. **Decide** assigns explicit ownership for acceptance, revision, or rejection. **Learn** feeds delivery metrics, incidents, review findings, and retrospective observations back into the team's working agreements as depicted in Table 2.

Area	GenAI assistance	Human control	Evidence / metric
Product	Research synthesis, problem framing, experiment variants	Customer meaning, prioritization, trade-offs	Decision quality, experiment outcomes
Planning	Story drafts, dependency summaries, acceptance criteria	Sprint goal and scope	Planning rework, carry-over
Engineering	Code, refactoring, explanations, migration drafts	Architecture, correctness, maintainability	Cycle time, defects, review effort
Testing	Unit tests, edge cases, failure explanations	Test intent and release confidence	Coverage, escaped defects
Security	Threat prompts, secure-code suggestions	Risk acceptance and controls	Findings, remediation time
Release	Notes, change summaries, runbook drafts	Go/no-go and operational ownership	Change failure, rollback

Table 2. Suggested operating model for controlled GenAI adoption in Agile teams.

X GOVERNANCE AND MEASUREMENT

Governance should be embedded into normal Agile routines rather than treated as a separate compliance exercise. Teams can define approved tools, prohibited data classes, review requirements, and escalation paths in working agreements. NIST AI RMF offers a useful governance vocabulary, while Scrum's inspect-and-adapt cycle provides a practical cadence for reviewing whether AI use is producing the intended outcomes as shown in figure 7 [1,14].

A. Measurement Model

A balanced measurement model should track at least five dimensions: delivery flow, engineering quality, developer experience, product outcomes, and risk. Examples include cycle time, deployment frequency, change-failure rate, escaped defects, review latency, perceived cognitive load, customer-reported outcomes, security findings, and policy exceptions. Metrics should be interpreted together because optimizing one dimension can damage another [22].

XI DISCUSSION

A. From Production to Validation

The most important organizational shift may be a change in the relative cost of production versus validation. When code, documentation, tests, and summaries become cheaper to draft, the scarce resource becomes the team's capacity to verify whether those artifacts are correct and valuable.

Agile teams should therefore strengthen feedback mechanisms at the same time that they adopt generation capabilities.

B. Engineering Management Implications

Engineering managers should avoid measuring GenAI adoption through raw code volume or prompt counts. The more meaningful questions are whether teams resolve work faster without increasing defects, whether review effort remains sustainable, whether developers gain or lose learning opportunities, and whether architecture remains coherent. DORA's recent research characterizes AI as an amplifier of existing organizational strengths and weaknesses, reinforcing the importance of the underlying delivery system [17].

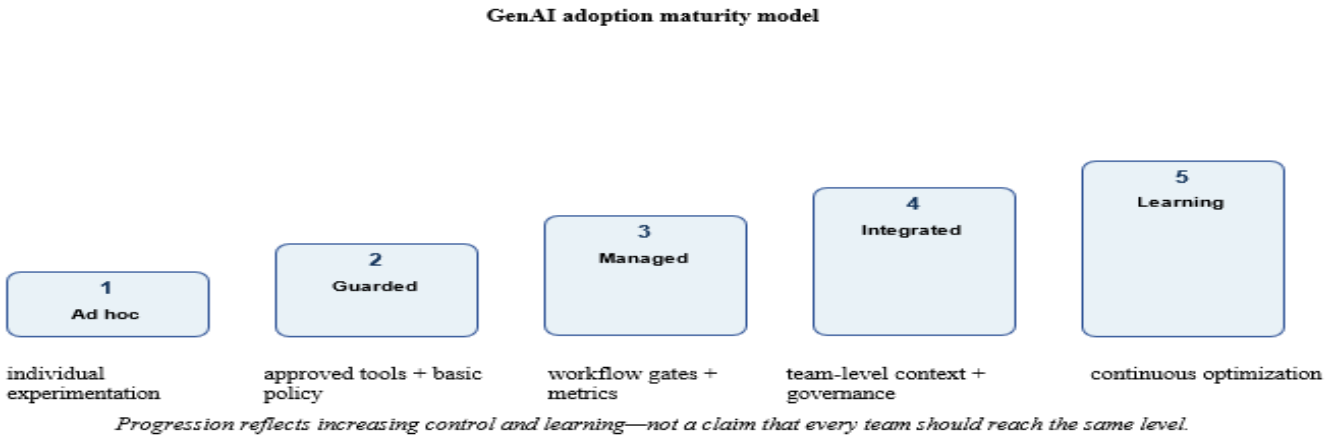

Fig. 7. GenAI adoption maturity model. The levels describe increasing integration and control, not a universal target

B. Suggested Measurement Dashboard

Dimension	Leading indicators	Lagging indicators	Interpretation
Flow	time-to-first-draft, review queue	cycle time, throughput	Did assistance reduce friction or mo
Quality	test generation, static findings	escaped defects, change failure	Did faster generation preserve reliab
People	tool adoption, flow interruptions	satisfaction, cognitive load	Did work become easier and sustain

Table 3. Balanced Measurement Model for GenAI adoption

C. Product Management Implications

For product leaders, the opportunity is to compress the distance between information and action: customer feedback can be synthesized more quickly, alternatives can be explored earlier, and technical explanations can be made more accessible. The corresponding risk is that teams may generate more ideas than they can validate. Product governance should therefore protect evidence-based prioritization and explicit hypotheses [21-23].

D. Agile Leadership Implications

Agile leadership increasingly involves designing the conditions under which humans and AI can work together safely. Working agreements should define where AI is

encouraged, where it is restricted, what must be reviewed, and how incidents are handled. Retrospectives can examine not only whether AI saved time but also whether it changed decision quality, collaboration, learning, and technical risk.

E. Limitations

The evidence base remains heterogeneous. Controlled experiments often measure narrow tasks, practitioner surveys measure perceptions, and rapidly changing tools make longitudinal comparisons difficult. The framework proposed here is a synthesis and operating model, not a claim that one adoption pattern has been experimentally proven superior across all Agile organizations. Future research should examine longitudinal team-level outcomes, review workload, security incidents, product metrics, and differences across domains and levels of AI maturity.

XII CONCLUSION

GenAI is changing Agile software and product work by lowering the cost of producing many intermediate artifacts: code, tests, explanations, requirements, summaries, and alternatives. Current evidence supports meaningful task-level acceleration and widespread practitioner adoption, while also showing that the effects of AI are broader than coding speed alone. The central organizational challenge is that faster generation can move constraints downstream into validation, review, integration, architecture, security, and decision-making.

A sustainable approach is therefore to place GenAI inside an explicit Human-in-the-Loop operating model. Teams should frame work carefully, use AI to generate candidate outputs, verify those outputs according to risk, preserve human decision ownership, and learn from measured outcomes. In this model, GenAI is neither treated as an autonomous team member nor reduced to a simple autocomplete tool. It is an adaptable assistant layer whose value depends on the quality of the Agile system surrounding it.

XIII REFERENCES

[1] Schwaber K, Sutherland J. The Scrum guide: the definitive guide to Scrum: the rules of the game [Internet]. Scrum.org & Scrum Inc.; 2020 Nov. Available from: https://www.scrumguides.org/scrum-guide.html

[2] Forsgren, N., Storey, M.-A., Maddila, C., Zimmermann, T., Houck, B., & Butler, J. (2021). The SPACE of Developer Productivity: There's more to it than you think. ACM Queue, 19(1). https://doi.org/10.1145/3454122.3454124

[3] Chen M, Tworek J, Jun H, Yuan Q, Ponde de Oliveira Pinto H, Kaplan J, Edwards H, Burda Y, Joseph N, Brockman G, Ray A, Puri R, Krueger G, Petrov M, Khlaaf H, Sastry G, Mishkin P, Chan B, Gray S, Ryder N, Pavlov M, Power A, Kaiser L, Bavarian M, Winter C, Tillet P, Such FP, Cummings D, Plappert M, Chantzis F, Barnes E, Herbert-Voss A, Guss WH, Nichol A, Paino A, Tezak N, Tang J, Babuschkin I, Balaji S, Jain S, Saunders W, Hesse C, Carr AN, Leike J, Achiam J, Misra V, Morikawa E, Radford A, Knight M, Brundage M, Murati M, Mayer K, Welinder P, McGrew B, Amodei D, McCandlish S, Sutskever I, Zaremba W. Evaluating large language models trained on code. arXiv [Preprint]. 2021 arXiv:2107.03374 [cs.LG]. Available from: https://arxiv.org/abs/2107.03374

[4] Shani I, GitHub Staff. Survey reveals AI's impact on the developer experience [Internet]. San Francisco (CA): GitHub; 2023 Jun 13. Available from: https://github.co.in/blog/2023-06-13-survey-reveals-ai-impact-on-developer-experience

[5] Peng S, Kalliamvakou E, Cihon P, Demirer M. The impact of AI on developer productivity: Evidence from GitHub Copilot. arXiv [Preprint on the Internet]. 2023. Available from: https://arxiv.org/abs/2302.06590

[6]. Barke S, James MB, Polikarpova N. Grounded Copilot: How programmers interact with code-generating models. arXiv [Preprint]. 2022 arXiv:2206.15000 [cs.HC]. Available from: https://arxiv.org/abs/2206.15000.

[7] Moradi Dakhel A, Majdinasab V, Nikanjam A, Khomh F, Desmarais MC, Jiang ZM. GitHub Copilot AI pair programmer: Asset or liability? J Syst Softw. 2023; 203:111734. https://doi.org/10.1016/j.jss.2023.111734.

[8] Zhang B, Liang P, Zhou X, Ahmad A, Waseem M. Practices and challenges of using GitHub Copilot: An empirical study. arXiv [Preprint]. 2023 arXiv:2303.08733 [cs.SE]. https://doi.org/10.48550/arXiv.2303.08733.

[9] Zhou X, Liang P, Zhang B, Li Z, Ahmad A, Shahin M, Waseem M. Exploring the problems, their causes and solutions of AI pair programming: A study on GitHub and Stack Overflow. J Syst Softw. 2024;219:112204. https://doi.org/10.1016/j.jss.2024.112204

[10] Hou X, Zhao Y, Liu Y, Yang Z, Wang K, Li L, Luo X, Lo D, Grundy J, Wang H. Large language models for software engineering: A systematic literature review. ACM Trans Softw Eng Methodol. 2024;33(8):Art. 220, 79 p. https://doi.org/10.1145/3695988

[11] Robert JE, Ozkaya I, Schmidt DC. Transforming software engineering and software acquisition with large

language models. In: Moallem A, Degen H, Ntoa S, editors. Artificial intelligence and large language models: a scientific perspective. Boca Raton (FL): CRC Press; 2026. Chapter 7; p. 111–131.

[12] Umar MA, Lano K. Advances in automated support for requirements engineering: a systematic literature review. Requirements Eng. 2024;29(2):177–207. https://doi.org/10.1007/s00766-023-00411-0

[13] Luu QH, Liu H, Chen TY. Can ChatGPT advance software testing intelligence? A case study on metamorphic testing. SSRN Electronic Journal [Preprint on the Internet]. 2023 Jun 28; [3 p.]. Available from: https://ssrn.com/abstract=4493849. https://doi.org/10.2139/ssrn.4493849

[14] Tabassi, E. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1. https://doi.org/10.6028/NIST.AI.100-1.

[15] OpenAI, Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, et al. GPT-4 technical report. arXiv [Preprint]. 2024 arXiv:2303.08774 [cs.CL]. Available from: https://arxiv.org/abs/2303.08774

[16] Irani B, Sonawane R. Accelerate the DevOps process on AWS. In: Irani B, Sonawane R. Generative AI-powered assistant for developers: accelerate software development with Amazon Q Developer. Birmingham: Packt Publishing; 2024. p. 355–384. https://doi.org/10.5040/bci-0kgn.ch-016.

[17] Al Haque E. Towards trustworthy AI-assisted software development. In: 2025 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC); 2025 Oct 19–23; Raleigh, NC, USA. Piscataway (NJ): IEEE; 2025. p. 431–432. https://doi.org/10.1109/VL-HCC65237.2025.00062.

[18] Belani H, Vukovic M, Car Z. Requirements engineering challenges in building AI-based complex systems. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW); 2019 Sep 23–27; Jeju, South Korea. Piscataway (NJ): IEEE; 2019. p. 252–255. https://doi.org/10.1109/REW.2019.00051

[19] France SL. Navigating software development in the ChatGPT and GitHub Copilot era. Bus Horiz. 2024;67(5):649–661. https://doi.org/10.1016/j.bushor.2024.05.009

[20] GitHub. (2024). Survey: The AI wave continues to grow on software development teams. GitHub Research. Survey of 2,000 enterprise respondents across the United States, Brazil, India, and Germany.

[21] Budhewar AS, Patil BK, Tharayil AS, Bhosale S, Suthadevan S, Patel N, Budhewar AS, Bhole HV, Patil PG, Somwanshi SK, Andy A. Federated AI-driven digital twins in the healthcare metaverse: architectures, privacy, and clinical intelligence. In: Ben Othman S, editor. The convergence of the metaverse, AI, and federated learning in healthcare ecosystems. Hershey (PA): IGI Global Scientific Publishing; 2026. p. 217–250. https://doi.org/10.4018/979-8-3373-8889-2.ch008

[22] Andy A, Tharayil AS, Budhewar AS, Andy D, R PJ. Introduction: artificial intelligence as a catalyst for next-generation gene and cell therapy. In: Andy D, Budhewar A, Andy A, Tharayil A, editors. AI-driven innovations in genome, epigenome, and mitochondrial therapies. Hershey (PA): IGI Global Scientific Publishing; 2027. p. 1–24. https://doi.org/10.4018/979-8-2600-0161-5.ch001

[23] Sinaga SSM, Indra Z, Wijaya MR, Almahdi MG. The role of generative AI in the software development life cycle: a systematic literature review. J Artif Intell Eng Appl. 2026;5(3):4273–4278. https://doi.org/10.59934/jaiea.v5i3.2431

