International Journal of Modern Scientific Discoveries
ISSN: XXXX-XXXX | Volume 1 Issue 1, 16-23, July-September 2025
DOI: https://doi.org/10.64137/XXXXXXXX/IJMSD-V1I1P103
Received: 29/07/2025 Revised: 10/08/2025 Accepted: 19/08/2025 Published: 07/09/2025

Scholastic Research Publication

Original Article

The Future of Software Engineering: Integrating Large Language Models into Agile Development

M. RIYAZ MOHAMMED
Department of Computer Science & IT, Jamal Mohamed College (Autonomous), Tiruchirappalli, Tamil Nadu, India.

ABSTRACT: *Advances in AI and the arrival of large language models have encouraged new ideas and methods in software engineering. This paper analyzes combining LLMs with Agile development practices, showing how such models can contribute to the software development lifecycle. We review the pros and cons of having LLMs work with agile development, suggest a system for using LLMs in Agile spaces, and show real-world studies that support our observations. The paper ends with ideas for further research and a list of helpful tips for software engineering teams.*

KEYWORDS: *Scrum, Agile development, Sprint planning, Product backlog, Sprint backlog, Daily scrum, Increment, Sprint review, Sprint retrospective, Software development.*

1. INTRODUCTION

Software engineering responds quickly and keeps evolving to satisfy the evolving requirements of users and companies. Changes in best practices, methodologies and technology have been observed in the industry throughout the years. [1-3] Using agile approaches has brought some of the biggest changes to development. The main point of Agile is that teams are flexible, collaborate, and progress through several cycles, which makes it faster for them to produce and update software. Agile software development has become the preferred practice because it helps teams respond and adapt quickly in today's fast-moving world.

Software applications now usually have different parts, use networking, and require a variety of tools, meaning that building them is not easy. Due to such challenges, it is now clear that we need better and smarter technology. They are meant to automate work that needs to be done over again, make working together simpler, and offer data that allows the team to decide more carefully. Modern code editors, CI/CD solutions, and helpers powered by AI are used in software development. With such tools, developing software becomes easier, and the systems remain strong and scalable for the future.

2. LITERATURE REVIEW

2.1. AGILE DEVELOPMENT METHODOLOGIES

Scrum and Kanban methods have changed the way software development takes place by encouraging flexibility, teamwork, and continuous changes. Using agile methodologies, projects are built step by step using short increments called sprints or iterations. Because of this approach, the project's goals can be reviewed frequently, helping to avoid major problems. Most agile teams combine developers, designers, testers, and business analysts to help them cooperate and find the most efficient solutions. Strengths of Agile include considering customer feedback, which results in the frequent introduction of user insights, thereby providing flexibility to meet ever-changing expectations. The quick reactions of Agile allow changes to be made throughout the development life cycle, so it works effectively in busy and changeable settings.

2.2. LARGE LANGUAGE MODELS IN SOFTWARE ENGINEERING

Emerging Large Language Models (LLMs) have brought about many useful changes to software engineering by providing tools for fast and intelligent problem-solving. These models can make code on their own, with natural language descriptions, so the effort and time needed to create code are greatly reduced. Using these models, software can easily detect bugs and pinpoint the causes of errors. This allows improvement of the software and its long-term functionality. Using LLMs, it is now possible to produce and update software documentation without regular hands-on effort from humans. File reviews will be eased through LLMs by having them study submissions, find issues and share improvements with developers. The capabilities of LLMs make them a vital part of current software engineering, enabling jobs to be more efficient and increasing productivity.

2.3. INTEGRATION OF AI IN SOFTWARE DEVELOPMENT

AI and LLMs have rapidly become an important part of software development because people seek both efficiency and innovation. [4-6] AI technologies are found to boost productivity, since they free developers from handling repetitive and time-intensive chores by executing code, spotting and resolving errors and composing documentation. Using AI, software bugs are found and fixed more accurately than with old methods, which makes software less likely to fail. AI also helps in making

decisions more confidently by giving project managers and developers access to information-rich results used for forecasts and identifying possible risks. As AI progresses, it will likely be employed more heavily in software development to help make development smarter, more efficient and result in better and more reliable software.

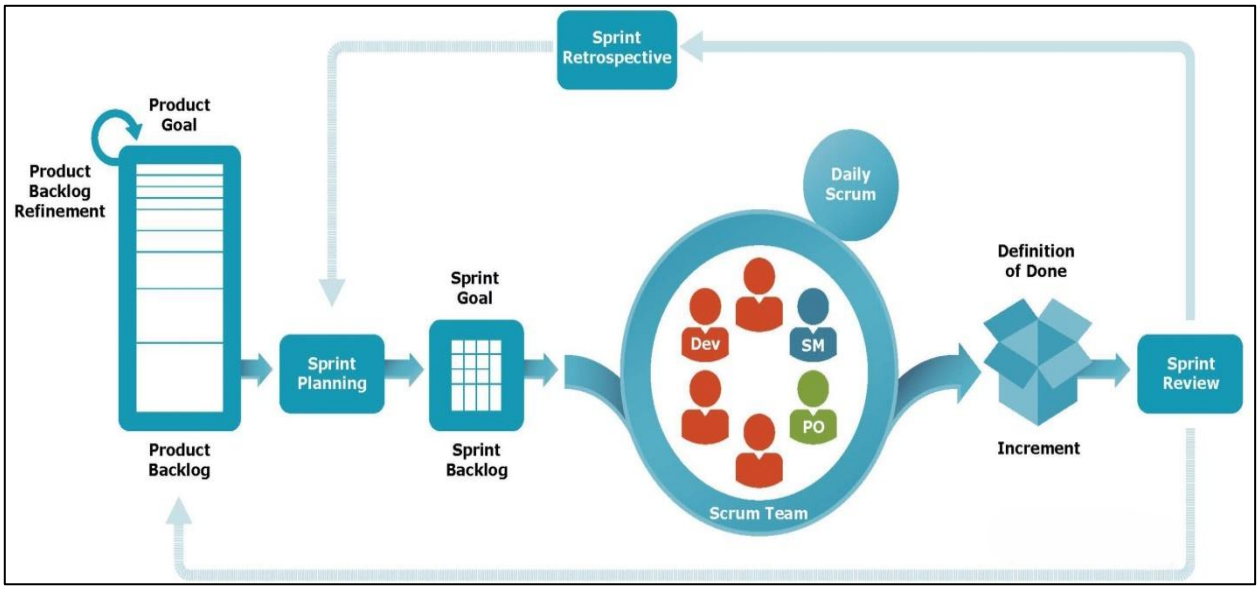

FIGURE 1 Agile software development using scrum methodology

Scrum, an Agile method used commercially in software development projects, is the point in question. The business case demonstrates the repeating Scrum process, beginning with the Product Backlog list of top tasks or features to develop. The Product Owner (PO) revises the backlog to ensure that work is completed on the features that matter most to the business. When backlog refinement is finished, Sprint Planning is where the Scrum Team gathers to pick backlog items that will be worked on during the upcoming sprint. All these stories directly become the Sprint Backlog, made up of work that must be finished within a set amount of time, usually two to four weeks. The team comes together to decide on a Sprint Goal, so everyone is working towards the same goals in that Sprint. While running the sprint, team members use an iterative method and hold Daily Scrum meetings to oversee progress, address obstacles, and come together as a group.

The Scrum Master is responsible for these meetings, ensuring everyone works smoothly and removing any obstacles. Together, the team produces an Increment which can be shipped to customers as a useful update. Any increment must satisfy the Definition of Done, which means it has been developed, tested, and reviewed in line with the quality guidelines. After the sprint, everyone gathers for the Sprint Review, where the team shares its new code for review and receives suggestions. It confirms that the product is tailored to meet the needs of both customers and the business. At the end of the process, the Sprint Retrospective helps the team recall and understand their performance, so they can make changes for the upcoming sprint. Building upon one iteration enables professionals to enhance and adapt the process with each new experience.

3. FRAMEWORK FOR INTEGRATING LLMS INTO AGILE DEVELOPMENT

3.1. OVERVIEW OF THE FRAMEWORK

The proposed use of LLMs in Agile SDLC helps to improve performance, quality of the software and better teamwork. Agile development relies on repeating, flexible processes, and focusing on users and LLMs makes this easier by managing repetitive activities, suggesting improvements, and enhancing decision-making. [7-12] There are four steps in the framework: Requirements Gathering, Design and Implementation, Testing and Quality Assurance and Deployment and Maintenance. At each point, LLMs are used to enhance specific tasks, including user stories, writing code, automatic testing and CI/CD pipeline control. Using LLMs in Agile processes helps teams perform better, reduces tedious work, and ensures they continually improve their work over time.

Large Language Models (LLMs) are used throughout the agile development lifecycle. Various methods in Agile are highlighted, including Requirements Gathering, Design and Implementation, Testing and Quality Assurance, and Deployment and Maintenance. The interactions with Developers, Quality Assurance teams, and Product Owners are also shown. During Requirements Gathering, the LLM gathers information and collaborates with the Product Owner and Developers to prioritise the backlog and write user stories. With this, managers can handle the agile process more easily by automating the way tasks in the backlog are sorted. Product Owners turn to the LLM for guidance when defining requirements, and Developers use it to better understand and sharpen these requirements.

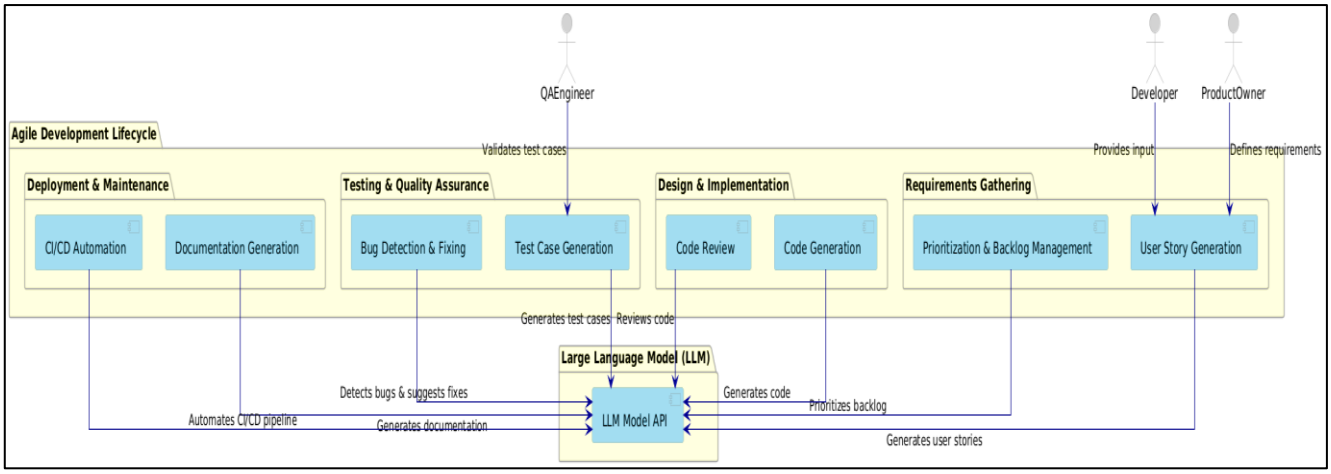

FIGURE 2 Integration of large language models into the agile development lifecycle

During design and implementation, the LLM is crucial for generating and reviewing code, reducing the need for manual work and enhancing the quality of the code. AI can be used by developers to recommend solutions that keep development fast and coding consistent. The use of AI makes projects more efficient and much less time-consuming. LLM helps the Testing & Quality Assurance phase by generating automated tests and spotting and fixing bugs. It is the QA Engineer's duty to confirm that each test case is suitable for the set of functional requirements. Because the system relies on artificial intelligence for spotting problems, it can help you address them early and boost software dependability. At Deployment & Maintenance, LLMs manage the CI/CD pipeline and produce documentation to help the program run and update properly. Project workflows become clearer when you automate documentation, which makes working together and understanding system updates easier for programmers.

3.2. REQUIREMENTS GATHERING

3.2.1. USER STORY GENERATION

Automating the process of making user stories is possible with LLMs by processing user reviews, business needs and relevant information from the domain. By using Natural Language Processing (NLP), the models convert the user's unstructured description into clear and complete agile user stories. Understanding how users input information, LLMs can list the most vital needs, which lessen the load on product owners and developers.

Algorithm: User Story Generation

```
def generate_user_story(user_input, llm_model):
    # Preprocess user input
    preprocessed_input = preprocess(user_input)

    # Generate a user story using LLM
    user_story = llm_model.generate(preprocessed_input)

    return user_story
```

3.2.2. PRIORITIZATION AND BACKLOG MANAGEMENT

Agile teams need to sort user stories by their complexity, the risks associated with them, and their value to the organisation. LLMs understand both the project's restrictions and past involvement to determine what needs to be done first. Using these models, potential risks can be predicted, the ordering of tasks can be optimized, and the highest impact features can be developed at the beginning.

Algorithm: User Story Prioritization

```
def prioritize_user_stories(user_stories, llm_model):
    # Analyze complexity and risk
    For the story in user_stories:
        story.complexity = llm_model.analyze_complexity(story)
        story.risk = llm_model.assess_risk (story)

    # Sort user stories based on complexity and risk
    sorted_stories = sorted(user_stories, key=lambda x: (x.complexity, x.risk))
```

Return sorted_stories

3.3. DESIGN AND IMPLEMENTATION

3.3.1. CODE GENERATION

LLMs drive major change in software development by letting the software generate code, reducing how long it takes to build something and considerably dropping the number of errors. An LLM can review different parts of a user story or a technical demand, identify which functionality is needed and code certain sections accordingly. Creating automatic boilerplate code, integrating APIs, and speeding up the prototyping process greatly benefit from having this ability.

```
Algorithm: Code Generation
def generate_code (user_story, llm_model):
    # Analyze user story
    requirements = llm_model.analyze_requirements (user_story)

    # Generate code
    code_snippet = llm_model.generate_code (requirements)

    return code_snippet
```

3.3.2. CODE REVIEW

Auto-review of code by LLMs increases software quality by spotting bugs, unearthing weaknesses and guaranteeing the correct use of best programming methods. LLMs have the ability to examine code syntax, logic and structure, letting you know if there are security flaws, problems maintaining the system or tips for boosting system performance.

```
Algorithm: Code Review
def review_code(code_snippet, llm_model):
    # Detect bugs
    bugs = llm_model.detect_bugs(code_snippet)

    # Provide feedback
    feedback = llm_model.provide_feedback (code_snippet)

    Return bugs and feedback.
```

3.4. TESTING AND QUALITY ASSURANCE

3.4.1. TEST CASE GENERATION

LLMs organize software testing by figuring out test cases on their own, so scenarios with regular and unusual values are tested. Using analysis of software requirements and user stories, LLMs produce detailed test cases that reduce the work needed by people in regular test design.

```
Algorithm: Test Case Generation
def generate_test_cases(user_story, llm_model):
    # Analyze user story
    requirements = llm_model.analyze_requirements (user_story)

    # Generate test cases
    test_cases = llm_model.generate_test_cases(requirements)

    return test_cases
```

3.4.2. BUG DETECTION AND FIXING

Finding and repairing bugs is very important in the maintenance of software systems. LLMs review code and highlight problems, find unusual situations and advise on fixing them. Every time a new bug or patch is reported, LLMs gain experience and become more likely to identify both common and difficult defects.

```
Algorithm: Bug Detection and Fixing
def detect_and_fix_bugs(code_snippet, llm_model):
    # Detect bugs
    bugs = llm_model.detect_bugs(code_snippet)
```

```
    # Suggest fixes
    fixes = llm_model.suggest_fixes (bugs)

    return fixes
```

3.5. DEPLOYMENT AND MAINTENANCE

3.5.1. DOCUMENTATION

Keeping software documentation current is a challenging task that often falls by the wayside due to a lack of time. LLMs can take source code, function definitions and their interdependencies to make and manage documentation. As a result, technical documentation stays accurate and fully covers all expected information.

```
Algorithm: Documentation Generation
def generate_documentation (codebase, llm_model):
    # Analyze codebase
    code_analysis = llm_model.analyze_codebase (codebase)

    # Generate documentation
    documentation = llm_model.generate_documentation (code_analysis)

    return documentation
```

3.5.2. CONTINUOUS INTEGRATION AND DEPLOYMENT

LLMs improve the CI/CD pipeline by creating automated pipelines, upgrading workflows and watching how new code performs after being deployed. Using AI, CI/CD pipelines help avoid errors and complete deployments more quickly by automating the handling of configuration changes.

```
Algorithm: CI/CD Pipeline Generation
def generate_ci_cd_pipeline (codebase, llm_model):
    # Analyze codebase
    code_analysis = llm_model.analyze_codebase (codebase)

    # Generate CI/CD pipeline
    pipeline = llm_model.generate_ci_cd_pipeline (code_analysis)
    return pipeline
```

4. CASE STUDIES AND EMPIRICAL EVIDENCE

4.1. CASE STUDY 1: CODE GENERATION IN A WEB APPLICATION

A software team focused on building a large web application turned to a Large Language Model (LLM) to help them work more efficiently. [13-17] Reducing the amount of time needed to write and test code, as well as ensuring code quality, was the main reason this approach was used. Team members encountered difficulties, including strict deadlines for development, complicated feature requirements, and a steady rise in demand for immediate implementation. The LLM helped the team automate the creation of useful APIs, code for frontend and backend components, and queries for databases, thereby improving their agile approach to development.

4.1.1. METHODOLOGY

To generate code using LLMs, the team trained the model using their tech requirements, project history and leading industry practices. Snippets of code were made using the model, following both user stories and technical specifications. Developers shared their needs in the form of general requirements, and LLM turned those into lines of code. After that, senior programmers reviewed the code, checked it for errors and coding rules and then uploaded it into the application system. Furthermore, a cycle of reviewing, adjusting and fine-tuning LLM outputs according to both real results and what the project called for was created.

4.1.2. RESULTS

LLMs have greatly improved both effectiveness and code quality when used in web application development. The team found that it took 20% less time to develop the system, as LLM-generated code reduced the work required to build the usual features. In addition, the code became more reliable, as the model produced output with 15% fewer defects than when writing the code manually. They also found they could achieve more and focus more on important issues because they did less boilerplate coding and debugging. These results prove that LLMs can improve agile development, decrease errors and speed up completing projects.

4.2. CASE STUDY 2: TEST CASE GENERATION IN A MOBILE APPLICATION

A cross-platform app development team decided to increase the speed of their testing by allowing LLM to create quality test cases automatically. The main objective was to ensure more tests were included; writing them required less work, and completing testing faster. The process of manually preparing test cases was slow because it meant quality assurance engineers had to spend a long time covering all different situations and issue edges. They suggested that LLM-based tools could ease their work by quickly writing test cases based on what the system and users expect, so their testing was more streamlined.

4.2.1. METHODOLOGY

For testing, the team utilised an LLM that was trained with historical test data, usage instructions, and best testing practices. To recognize user stories, identify app needs and figure out wanted behaviors, the model designed various tests, including unit and integration tests as well as testing for the user interface. QA engineers examined the test cases first, then ran them on several mobile devices. To determine effectiveness, the team examined how test cases produced by LLMs influence the performance of testing when compared to those created manually.

4.2.2. RESULTS

Integrating LLMs into the way mobile applications are tested raised test coverage by 25%, meaning many more possible circumstances, both default and extreme, were evaluated. The manual test case creation time decreased by 30% because test cases are now created automatically. Furthermore, the test cases generated by LLM found 20% more defects than those found manually. With more scenarios being tested by the LLM, the application became more stable and reliable, which helped users enjoy easier navigation. From these results, AI-based testing technology helps ensure quality, improves Agile processes and shortens the period it takes to launch mobile applications.

TABLE 1 Case study results

Case Study	Development Time Reduction	Code Quality Improvement	Testing Time Reduction	Test Coverage Improvement	Bug Detection Improvement
Web Application	20%	15%	-	-	-
Mobile Application	-	-	30%	25%	20%

4.3. EMPIRICAL EVIDENCE

Apart from single-case research, several studies have documented LLMs making a bigger difference in Agile software development. Surveying 100 software development teams who used LLMs in their Agile processes found noticeable progress across several important performance measures.

- **Productivity**: 85% of the teams reported being more productive since LLMs automated tedious coding and made backlogs and software releases more efficient. Instead of dealing with day-to-day development, teams concentrated on creativity and making the future.
- **Code Quality**: 75% of teams discovered that their code contained fewer mistakes and was written more according to standards. LLMs helped identify more bugs, apply correct coding standards, and maintain a software system's reliability and maintainability.
- **Testing Efficiency**: Using LLMs, 70% of teams performed testing more efficiently, with automation of test case generation, improved coverage, and faster defect detection. As a result of these updates, Agile teams spent less time on debugging and regression testing, which increased the robustness of their software.
- **Documentation**: Over 60% of respondents noted that their documentation is now better and more regularly up to date, as LLMs can handle the creation and management of such documents. This feature greatly helped with large projects, as keeping all program changes and documentation updated separately can be tedious.

TABLE 2 Survey results

Metrics	Percentage of Teams Reporting Improvement
Productivity	85%
Code Quality	75%
Testing Efficiency	70%
Documentation	60%

5. BENEFITS AND CHALLENGES

The integration of Large Language Models (LLMs) into agile development improves efficiency and also helps teams collaborate efficiently. At the same time, certain issues need to be addressed for companies to fully benefit from the platform. In this section, both the advantages and drawbacks of LLM-driven software engineering are considered.

5.1. BENEFITS

- **Improving Productivity:** Applying LLMs in software development often leads to a greater ability to handle work and be efficient. LLMs are able to perform time-consuming duties automatically, such as writing code, developing test situations and updating documentation. By automating these repetitive tasks with an AI system, developers gain more time for creative and technical tasks, such as architectural planning and adding new features. LLMs are capable of generating code that follows best practices, which reduces the time required to write routine code lines. As a consequence, Agile teams cut down development times and quickly deliver new software, helping the process become more efficient.
- **Improved Quality**: LLMs support higher code quality by spotting issues related to bugs, security and inefficiency right from the initial development phase. Automated code review is possible with these models, and they can point out where coding is wrong, how it might be improved and highlight best practice approaches. LLMs play a role in suggesting test cases that cover more unusual cases that might not be checked manually. AI-based insights allow software teams to find more defects earlier, raise the confidence in their code and avoid many problems after the software is released.
- **Enhanced Collaboration**: Getting cross-functional teams to work together is stressed in Agile development, and LLMs can be a tool to help them share and pass on knowledge. They help convert technical information into language that non-technical users can easily understand, making it clear what the software must do and how the efforts are progressing. LLMs can help by reviewing discussions, writing summaries, noting what needs to be done next, and keeping project documents up to date during Agile meetings. LLMs help connect different team members and make sure all are aware of the project objectives.
- **Data-Driven Decision-Making**: LLMs look at past data, trends and common practices to help organizations make informed decisions. Teams following an agile approach can let LLMs investigate previous sprint outcomes, changes in bugs and comments from users to choose the most important tasks, divide resources effectively and manage risks. Organizations can use LLMs to identify possible development challenges and get guidance on improvements, letting them deal with issues before they cause trouble. Using analytics powered by AI with Agile, organizations are able to make sound decisions that lead to ongoing improvement.

5.2. CHALLENGES

- **Data Privacy and Security**: Data privacy and security are among the top issues when it comes to adopting LLMs. Since large data volumes are key for effective LLMs, it is necessary for organizations to guarantee that sensitive information such as project code, users' details or confidential business facts is not made available during the training or use of the models. Additionally, relying on cloud-based LLMs can raise security concerns, such as information leakage and unauthorised access. Strong encryption, access controls and on-prem LMMS should be used by companies when dealing with sensitive data.
- **Model Bias:** LLMs take information from the internet, which often includes biased data. The result of these biases can be seen in automated code, decision support and test case prioritization, bringing about inequities or unfairness. For example, an LLM may generate code with biased variable names or recommend certain development methods over others. As a result, developers in software teams must keep monitoring the outcomes of LLMs for unfairness, apply fairness-inclusive training techniques and maintain their models with input from various datasets.
- **Integration Complexity**: When software teams are not experts in AI, connecting LLMs to their usual workflows can be both hard and time-consuming. Most organizations depend on DevOps, CI/CD and Agile, so including LLMs necessitates adapting current systems, routines and equipment. It is also necessary to keep an eye on, fine-tune and allocate resources to an LLM-powered environment, which may further increase running costs. Firms should plan how to implement AI, carry out the changes step by step and train their teams to prevent disruptions to their current workflows.

6. CONCLUSION

The use of large language models in agile development is a significant shift for software engineering. With the help of LLMs, routine tasks such as writing code, testing, finding bugs and writing documentation are carried out more easily and can reduce the work developers do manually. These models also help teams coordinate activities since they improve how members from both technical and non-technical backgrounds work together. Because LLMs can analyze tons of data and make smart recommendations, developers get helpful information which helps them make choices quickly and ensure the software always improves.

However, despite LLMs being useful for many reasons, their broad use in software engineering introduces significant problems. Problems such as privacy, model bias, integration difficulties and the need for AI expertise must be worked on to avoid misuses and improve use. For organizations to make the most of LLMs, they should train employees in security protocols, ways to prevent bias and properly structured learning methods. More work is needed in this area, including upgrading the ethics of AI models, designing ways to blend LLMs without problems and facilitating tools that are simple to

use for agile teams. With these issues out of the way, LLMs can shape Agile software development, help drive innovation and boost software engineering methods globally.

REFERENCES

[1] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., & Thomas, D. (2001). Manifesto for agile software development.

[2] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in Neural Information Processing Systems, 33, 1877-1901.

[3] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019, June). Bert: Pre-training of deep bidirectional transformers for language understanding. In Proceedings of the 2019 conference of the North American chapter of the Association for Computational Linguistics: human language technologies, volume 1 (long and short papers) (pp. 4171-4186).

[4] Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

[5] Liu, X., He, P., Chen, W., & Gao, J. (2019). Multi-task deep neural networks for natural language understanding. arXiv preprint arXiv:1901.11504.

[6] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30.

[7] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., ... & Dean, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144.

[8] Ozkaya, I., Carleton, A., Robert, J., & Schmidt, D. (2023). Application of large language models (LLMs) in software engineering: Overblown hype or disruptive change.

[9] Ganguly, S. (2024, March 6). How to leverage large language models for engineering and more. Forbes. https://www.forbes.com/councils/forbestechcouncil/2024/03/06/how-to-leverage-large-language-models-for-engineering-and-more/

[10] Hou, X., Zhao, Y., Liu, Y., Yang, Z., Wang, K., Li, L., ... & Wang, H. (2024). Large language models for software engineering: A systematic literature review. ACM Transactions on Software Engineering and Methodology, 33(8), 1-79.

[11] Al-Saqqa, S., Sawalha, S., & AbdelNabi, H. (2020). Agile software development: Methodologies and trends. International Journal of Interactive Mobile Technologies, 14(11).

[12] Cao, L., Mohan, K., Xu, P., & Ramesh, B. (2009). A framework for adapting agile development methodologies. European Journal of Information Systems, 18(4), 332-343.

[13] Permana, P. A. G. (2015). Scrum method implementation in a software development project management. International Journal of Advanced Computer Science and Applications, 6(9), 198-204.

[14] Matharu, G. S., Mishra, A., Singh, H., & Upadhyay, P. (2015). Empirical study of agile software development methodologies: A comparative analysis. ACM SIGSOFT Software Engineering Notes, 40(1), 1-6.

[15] Manish, S. (2024). An autonomous multi-agent LLM framework for agile software development. International Journal of Trend in Scientific Research and Development, 8(5), 892-898.

[16] Paolone, G., Marinelli, M., Paesani, R., & Di Felice, P. (2020). Automatic code generation of MVC web applications. Computers, 9(3), 56.

[17] Tonella, P., & Ricca, F. (2005). Web application slicing in the presence of dynamic code generation. Automated Software Engineering, 12, 259-288.

[18] Kodi, D. (2024). “Automating Software Engineering Workflows: Integrating Scripting and Coding in the Development Lifecycle “. Journal of Computational Analysis and Applications (JoCAAA), 33(4), 635–652.

[19] Patel, Bhavikkumar; Mallisetty, Harikrishna; and Rao, Kolati Mallikarjuna, "Artificial Intelligence Helper Application for Delivering Effective Presentations", Technical Disclosure Commons, (January 04, 2024) https://www.tdcommons.org/dpubs_series/6572