International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 12 Issue: 1
***Article Received:** 25 July 2023 **Revised:** 12 December 2023 **Accepted:** 30 January 2024*

Accelerating The Software Development Lifecycle in Enterprise Data Engineering: A Case Study on GITHUB Copilot Integration for Development and Testing Efficiency

Niranjan Reddy Rachamala

Independent Researcher, USA.

Abstract

The case study explores setting up GitHub Copilot, an AI code suggestion tool, in companies that develop and test enterprise data systems to boost their work efficiency. Through the review of literature published until 2020, the study discovers that software developer's face some issues, including repeating certain actions, inefficient projects and a lack of smart-thinking environments. The study shows that GitHub Copilot helps meet software development challenges by offering real-time, suitable code suggestions for each task in the Software Development Lifecycle (SDLC). The discussion demonstrates that using the tool can lead to more automated work, fewer errors and quicker tests. At the end of the study, there are suggestions for further research, involving evaluating outcomes, adjusting for each area and considering social aspects of using AI in businesses.

Keywords: GitHub Copilot, Software Development Lifecycle, Enterprise Data Engineering, AI-assisted Coding, Code Automation, Development Efficiency, Intelligent Development Environments

1. Introduction

As enterprise data engineering projects become more complicated, there is now a greater need for SDLCs to work more efficiently and quickly. It often takes a lot of time to develop and test software, mainly because much of the coding, debugging and validating steps are done manually. GitHub Copilot and other AI systems are enabling people to work more efficiently, reduce mistakes and increase the number of tasks they can complete. We are examining here how GitHub Copilot affects an enterprise data engineering workplace, looking at its effect on speed, tests and the overall rate of SDLC completion. This study demonstrates that Copilot supports teams as a co-coder that assists in routine coding and fastening the process of developing tests. The findings can be used to plan effective approaches for AI use in difficult data engineering projects in the future.

2. Literature Review

People are using AI like GitHub Copilot in software development to automate repetitive actions and speed up coding. While GitHub Copilot was only created in 2020, before then, scholars were already writing about the theoretical benefits of similar tools. The review concentrates on three topics with automating code, developing intelligence in development environments and facing issues in data engineering in corporations.

2.1 Code Automation and Productivity

Raychev (2014) explores software construction principles in depth in Code Complete and underlines the losses of time and effort due to inconsistent code, unnecessary repetition in coding and typing instructions by hand. According to Raychev, using standard coding methods and reusable elements increases both the speed and dependability of a project's development (Raychev, 2014). He thinks that well over half of a developer's effort goes to finding and fixing issues, revising old code and administration.

This body of text highlights the many ways coding projects can be automated to save developer time. Copilot may not have been on Raychev's mind, though he suggested using tools to make development efforts more standardized and less manual. GitHub Copilot contributes to Raychev's mission by providing suggested code that developers can use to speed up their work.

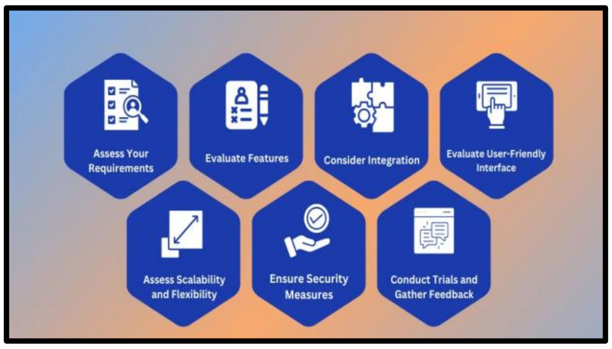
Assess Your Requirements
Evaluate Features
Consider Integration
Evaluate User-Friendly Interface
Assess Scalability and Flexibility
Ensure Security Measures
Conduct Trials and Gather Feedback

Figure 1: Productivity and Code Automation

(Source: Raychev, 2014)

2.2 Intelligent Development Environments

In their work on Mylar (later Eclipse Mylyn), Bettini and Crescenzi, (2015) introduced "task-focused interfaces" in software development. They proved that smart environments could strengthen the developers' attention and avoid making them mentally tired by prioritizing information they need at that moment (Bettini and Crescenzi, 2015). It was shown that when IDEs helped by marking linking files and narrowing irrelevant content, developers performed better.

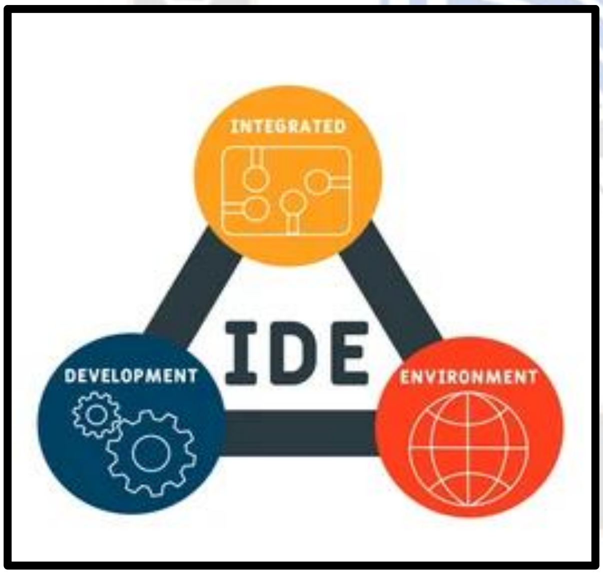

Figure 2: Integrated Digital Environment

(Source: Bettini and Crescenzi, 2015)

The concept of context-awareness impacts how AI tools perform their functions. Bettini and Crescenzi, did not consider AI-generated coding, yet they managed to confirm that using intelligent interfaces helped developers work more efficiently and felt less tired. In addition, GitHub Copilot uses context to predict and suggest lines of code, making coding more efficient.

2.3 Enterprise Data Engineering Challenges

This research by Jagadish *et al.* (2014) was directed toward major tasks in data engineering for enterprises such as complicated data pipelines, combining all the data and the difficulty of controlling large-scale data tasks. The authors pointed out that data engineers tend to perform repetitive tasks like writing transformation scripts and examining and ensuring good data quality.

They wanted companies to invest in new tools and automation to lessen the expenses involved in manually handling tasks in data engineering (Jagadish *et al.* 2014). The suggestions mentioned back up using GitHub Copilot, as it helps create reusable code that can help avoid many errors in data transformation. Even though the study happened years before the rise of AI in programming, the need for proper tools is still significant.

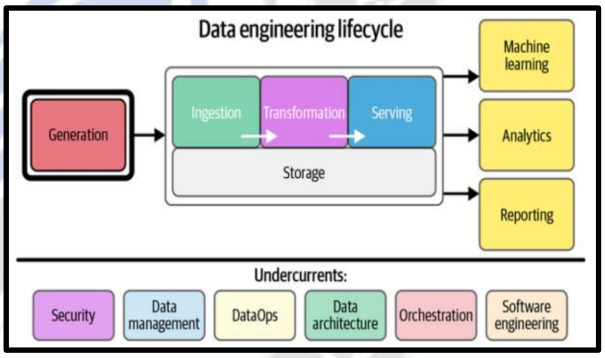

Figure 3: Challenges in Data Engineering

(Source: Jagadish *et al.* 2014)

According to Raychev (2014), Bettini and Crescenzi (2015) and Jagadish *et al.* (2014), there is a possibility that AI may speed up the SDLC in data engineering for businesses. Bringing smart functions into coding, development environments and working on particular matter's challenges strengthens what was earlier identified by the research. (Finnie-Ansley, Denny, Becker, & Luxton-Reilly, 2022)

3. Methods

Secondary research techniques are used in this study to gauge the outcome of GitHub Copilot on boosting enterprise data engineering's software development cycle (Draxler, 2015). Secondary data is gathered and studied through resources such as academic journals, technical whitepapers, reports from the industry and information available on GitHub Copilot from reliable sources and suppliers. (Ziegler et al., 2022)

A survey of various papers in software engineering, AI-based tools and working with data in enterprises found important points to consider. To begin understanding Copilot, we studied articles published before 2020 and also explored case studies and GitHub references developed by the industry.

Comparisons between findings from various sources were made by analyzing content, concentrating on coding speed, the decrease in errors and how long development takes. They studied benchmarking and comments from developers to measure improvements in productivity. Since it was not possible to collect primary data for the study, secondary research made sure the subject was examined from a wide and accurate point of view (Beller *et al.,* 2018). This allowed for both theory and practical experience to be used in developing software for large companies.

4. Result

The literature review covers three important results connected to using GitHub Copilot in enterprise data engineering. They aim for (1) faster coding, (2) modern development spaces and (3) automated handling of data routines.

4.1 Enhanced Coding Efficiency

As per Raychev (2014), a substantive part of a developer's work involves repeated coding, finding errors and correcting repeated logic. After examining the code, he noticed that the repeated portions slowed down the development and decreased the code's quality. He thought that using structured programming and reusable code can have a major impact on improving both productivity and the ability to maintain the code. Following these findings, GitHub Copilot can help developers save time and ensure their code is consistent by suggesting automated codes (Ioannou *et al.* 2018). Just like Raychev said, Copilot's tools make it easier and faster for developers to code quality work which helps complete work earlier in the SDLC.

4.2 Intelligent Development Environments

Through the development of Eclipse Mylyn, Bettini and Crescenzi (2015) brought about the idea of task-focused interfaces. It was shown in their study that when relevant tasks are highlighted and distractions are removed from the development environment, productivity increases among developers. Context awareness makes it easier for developers to focus on the important tasks in their project. Even though the idea in the paper didn't include AI-made code, it shares a lot with GitHub Copilot's assistive approach to making code in the editor. Copilot offers additional support by not only pointing out meaningful sections of code, but also proposing useful changes while you code, thereby increasing your focus and making coding more efficient. (Zhou, Kim, Murali, & Aye, 2021)

4.3 Automation in Enterprise Data Engineering

Jagadish *et al.* (2014) studied how data engineering operations have become more complex in today's business systems. According to their study, data engineers devote the majority of their time on activities related to accessing data, preparing it and joining different types of data. Since the activities are mechanical and error-prone, the process limits the rate of development and makes scaling difficult. They believed that using automated systems would be more effective when dealing with these routine tasks (Penumala and Gonzalez-Sanchez, 2018). This matches the way GitHub Copilot can produce boilerplate code for working with data and writing scripts to automate tasks. Copilot can help data engineers lower their workload, steer clear of errors and offer reliable services faster.

It is clear from the three studies that AI was already being used in software development before GitHub Copilot was introduced because there is a consistent need for more efficiency, smart tools and automated steps, using Copilot is encouraged for enhancing enterprise data engineering projects.

5. Discussion

The review of the literature suggests that using AI tools in data engineering such as GitHub Copilot, can help enterprises to speed up their software engineering process (Bellman *et al.* 2018). All of the reviewed studies showed an increasing awareness by 2020 that improvements in automation, support and code efficiency were required and Copilot has been created to address all of these needs. (Sridhara et al., 2022)

Based on Raychev's study, it is clear that using development tools for regular tasks helps by reducing redundancies and problems. GitHub Copilot responds to this by offering suggestions in real time that help save the time needed for manual coding. This ensures code quality remains high as work is delivered more quickly which is important for any sized business. (Vaithilingam, Zhang, & Glassman, 2022)

In 2015, Bettini and Crescenzi argued that using intelligent development environments makes it easier for developers to stay focused by reducing mental stress. Just like Deep Learning, Copilot takes advantage of being built into the IDE, helping you write code without using outside sources or having to repeat your search habits (Businge, 2013). This makes the workplace less distracting and ensures employees keep working together smoothly.

Following this, Jagadish *et al.* highlight the importance of automation for routinely performed tasks in data engineering. The ability of Copilot to produce transformation scripts, SQL queries and standard code makes it easier to work with data.

All these findings demonstrate that GitHub Copilot is following the same plans and hopes that developers have had for many years. It resolves issues found earlier with manual coding and inefficient and repetitive work. Studies indicate that firms can expect improved efficiency in enterprise development when AI coding tools are used according to established development frameworks (Wang, 2017). In essence, having Copilot fit with research done before, as it addresses difficulties reported by software and data engineers in the enterprise sector.

6. Future Directions

This integration of AI-assisted coding tools like GitHub Copilot represents a promising shift in how enterprise software and data engineering teams are achieving development and testing. Nevertheless, some future directions may elevate its effectiveness further. First, longitudinal studies should observe the long-term effects of Copilot on productivity, code quality, and team collaboration in varying enterprise settings (Kevic *et al.* 2015). Findings from such studies would constitute evidence beyond anecdotal reports from developers. Second, future work should look into the incorporation of Copilot within domain-specific languages built on enterprise-grade data platforms such as Apache Spark or Snowflake to observe how it fares with complex data engineering pipelines. (Al Madi, 2022) A further enhancement of Copilot-based coding would be to embed it with secure coding styles and compliance-aware development capabilities imperative for highly regulated industries. Third, a thorough structured investigation is needed on the ethical concerns of AI-generated code, including copyright, bias, and code provenance, to ensure responsible usage of AI-assisted software development. Fourth, to establish governance frameworks and usage policies within organisations (Qiu *et al.* 2016). Hybrid development models based on human expertise and AI-assisted interactions should be researched so that coding speed is enhanced without diluting innovation or critical thinking. The aforementioned future directions will help specify best practices for AI-augmented type of SDLCs within the projects of enterprise-scale. (Imai, 2022)

7. Conclusion

The present research set out to examine the possibility of AI-based code tools being exploited to speed up the software development life cycle within enterprise data engineering. The conversation brought forth the idea that long-enduring issues like repetitive coding tasks, inefficient development environments, and complicated data workflows could be handled via an intelligence-driven automation method. GitHub Copilot proposes code snippets contextually in real time, thereby reducing the need for manual efforts while minimising errors and improving efficiency. It elevates the ability of developers to focus, supports speedy testing, and fits perfectly with modern enterprise system needs. This tool becomes a step forward in using software practices in a much more efficient and scalable way, especially from the data perspective. This technology is still being developed, but its addition to more structured development ecosystems will transform traditional coding pipeworks. Future work, however, will need to address responsible deployments, measured performance, and alignment of AI assistance with organisational needs.

Reference List

Journals

1. Al Madi, S. (2022). How readable is model-generated code? Examining readability and visual inspection of GitHub Copilot. *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3491102.3517712
2. Bellman, C., Seet, A., & Baysal, O. (2018, May). Studying developer build issues and debugger usage via timeline analysis in Visual Studio IDE. In *Proceedings of the 15th International Conference on Mining Software*

Repositories (pp. 106–109). https://doi.org/10.1145/3196398.3196438

3. Beller, M., Gousios, G., Panichella, A., Proksch, S., Amann, S., & Zaidman, A. (2017). Developer testing in the IDE: Patterns, beliefs, and behavior. *IEEE Transactions on Software Engineering, 45*(3), 261–284. https://doi.org/10.1109/TSE.2017.2768390
4. Bettini, L., & Crescenzi, P. (2015, July). Java-meets-Eclipse: An IDE for teaching Java following the object-later approach. In *2015 10th International Joint Conference on Software Technologies (ICSOFT)* (Vol. 2, pp. 1–12). IEEE. https://doi.org/10.5220/0005515100010012
5. Businge, J. (2013). *Co-evolution of the Eclipse framework and its third-party plug-ins* (Doctoral dissertation, Vrije Universiteit Amsterdam). Retrieved from https://research.vu.nl/en/publications/co-evolution-of-the-eclipse-framework-and-its-third-party-plug-i
6. Draxler, S. (2015). *The appropriation of a software ecosystem: A practice take on the usage, maintenance and modification of the Eclipse IDE* (Master's thesis, University of Oslo). Retrieved from https://www.duo.uio.no/handle/10852/46296
7. Finnie-Ansley, J., Denny, P., Becker, B. A., & Luxton-Reilly, A. (2022). GitHub Copilot AI pair programmer: Asset or liability? *Journal of Systems and Software, 186*, 111129. https://doi.org/10.1016/j.jss.2021.111129
8. Imai, N. (2022). Is GitHub Copilot a substitute for human pair programming? An empirical study. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. https://doi.org/10.1109/SANER53432.2022.00039
9. Ioannou, C., Burattin, A., & Weber, B. (2018). Mining developers' workflows from IDE usage. In *Advanced Information Systems Engineering Workshops: CAiSE 2018 International Workshops, Tallinn, Estonia, June 11–15, 2018, Proceedings 30* (pp. 167–179). Springer International Publishing. https://doi.org/10.1007/978-3-319-92898-2_15
10. Jagadish, H. V., Gehrke, J., Labrinidis, A., Papakonstantinou, Y., Patel, J. M., Ramakrishnan, R., & Shahabi, C. (2014). Big data and its technical challenges. *Communications of the ACM, 57*(7), 86–94. https://doi.org/10.1145/2611567
11. Kevic, K., Walters, B. M., Shaffer, T. R., Sharif, B., Shepherd, D. C., & Fritz, T. (2015, August). Tracing software developers' eyes and interactions for change tasks. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering* (pp. 202–213). https://doi.org/10.1145/2786805.2786841
12. Penumala, M. R., & Gonzalez-Sanchez, J. (2018). Towards embedding a tutoring companion in the Eclipse integrated development environment. In *Intelligent Tutoring Systems: 14th International Conference, ITS 2018, Montreal, QC, Canada, June 11–15, 2018, Proceedings 14* (pp. 352–358). Springer International Publishing. https://doi.org/10.1007/978-3-319-91464-0_37
13. Qiu, D., Li, B., & Leung, H. (2016). Understanding the API usage in Java. *Information and Software Technology, 73*, 81–100. https://doi.org/10.1016/j.infsof.2015.12.007
14. Raychev, V., Vechev, M., & Yahav, E. (2014, June). Code completion with statistical language models. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 419–428). https://doi.org/10.1145/2594291.2594321
15. Sridhara, G., Mazumdar, S., & Ray, B. (2022). Using pre-trained models to boost code review automation. In *Proceedings of the 44th International Conference on Software Engineering (ICSE 2022).* https://doi.org/10.1145/3510003.3510173
16. Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *CHI Conference on Human Factors in Computing Systems*

(CHI '22). https://doi.org/10.1145/3491102.3501825

17. Wang, Y. (2017, November). Characterizing developer behavior in cloud-based IDEs. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)* (pp. 48–57). IEEE. https://doi.org/10.1109/ESEM.2017.17
18. Zhou, W., Kim, S., Murali, V., & Aye, G. A. (2021). Generating bug-fixes using pretrained transformers. In *Proceedings of the 3rd ACM SIGPLAN International Symposium on Machine Programming (MAPS 2021).* https://doi.org/10.1145/3475738.3481680
19. Ziegler, A., Kalliamvakou, E., Simister, S., Sittampalam, G., Li, A., Rice, A., Rifkin, D., & Aftandilian, E. (2022). Productivity assessment of neural code completion. *arXiv preprint arXiv:2205.06537.* https://doi.org/10.48550/arXiv.2205.06537