The impact of GitHub Copilot on developer productivity from a software engineering body of knowledge perspective

Danie Smit
University of Pretoria, danie.smit@bmwgroup.com

Hanlie Smuts
University of Pretoria, hanlie.smuts@up.ac.za

Paul Louw
University of Pretoria, paul.louw@bmwithub.co.za

Julia Pielmeier
BMW Group, julia.pielmeier@bmw.co.za

Christina Eidelloth
BMW Group, christina.eidelloth@bmw.de

The Impact of GitHub *Copilot* on Developer Productivity From a Software Engineering Body of Knowledge Perspective

Completed Research Full Paper

Danie Smit
University of Pretoria
d5mit@pm.me

Hanlie Smuts
University of Pretoria
hanlie.smuts@up.ac.za

Paul Louw
BMW IT Hub ZA
paul.louw@bmwithub.co.za

Julia Pielmeier
BMW Group
julia.jp.pielmeier@bmw.co.za

Christina Eidelloth
BMW Group
christina.eidelloth@bmw.de

Abstract

Recent times saw captivating improvements to artificial intelligence-assisted code completion technology. The precise effect this contributes to conventional software development is an exciting consideration for individuals and corporations seeking modern workflow optimization approaches. A team's productivity may be defined as the amount of work that can be completed in a certain amount of time. However, determining an exact metric for productivity proves to be a challenging task. To this end, a case study was conducted at a leading automotive organization investigating the effects on software developers who employ GitHub *Copilot* as part of their daily work. Typical agile working model metrics were evaluated in context of the SPACE framework to devise a means of measuring GitHub *Copilot's* impact on developer productivity. Some improvements were observed from the case study across the throughput, cycle time, code quality, defects, and developer satisfaction measures. This ultimately led to an increase in developer productivity.

Keywords

GitHub *Copilot*; software engineering; software developer productivity; artificial intelligence; SPACE framework

Introduction

The capabilities of artificial intelligence (AI) have recently extended beyond computational tasks toward generative AI. One example of generative AI is large language models (LLMs). LLMs are trained on large datasets and can generate text responses based on prompts. Through LLMs and transfer learning, AI agents can also generate code from natural language inputs (Chen et al. 2021). For instance, *CodeBERT* and *CodeT5* are open-source models based on *BERT* (Feng et al. 2020) and *T5* (Raffel et al. 2019) respectively. Another example is *Codex*, a model based on OpenAI's GPT-3 and trained on millions of GitHub repositories (Chen et al. 2021). This built the foundation for generative AI code-completion tools like GitHub *Copilot* (*Copilot*). As its name suggests, it was designed to collaboratively assist software developers with their tasks and is available as an extension of *Visual Studio Code* and *IntelliJ* integrated development environments (IDEs) (Moradi Dakhel et al. 2023). Within the IDE *Copilot* helps developers to write source code by offering inline code completion as well as a chat functionality that acts as an assistant that answers coding specific questions (Ziegler et al. 2022).

Copilot is advertised to increase developer productivity by reducing the amount of time required for coding and by helping developers to maintain focus. GitHub supports this claim with a commissioned study, which found that development tasks were completed 55% faster with *Copilot* (Kalliamvakou 2022). While the extent differs, the general assertion of *Copilot* increasing developers productivity is also supported by other, more independent sources. In an internal study from Accenture involving around 450 developers, the use of *Copilot* resulted in more throughput and higher quality from their developer teams (Salva et al. 2023). Similarly, a study by Mercedes-Benz reported an estimated increase in developer productivity of 30 minutes per day (Fahner and Gerth-Ramos 2023). This finding underscores the perceived productivity gains attributed to the adoption of *Copilot*. However, as other studies by Kalliamvakou (2022) and Salva et al. (2023) emphasize, Mercedes-Benz highlights that total efficiency gain is difficult to determine. Indeed, the effects of emerging generative AI technologies on software development are thought-provoking.

Therefore, this study explores: *"How does the integration of Copilot affect software developer productivity in an automotive organization's IT Hub?"*. The IT Hub is a distinct software development office where most of the organization's information technology (IT) development takes place. Currently, the demands for tasks are greater than the capacity. A case study research strategy was followed (Benbasat et al. 1987) to gain insights posed by this captivating question. This strategy allowed for a comprehensive in-house investigation to determine the productivity gain from *Copilot* utilization.

The upcoming sections first contextualize modern software development and productivity. Second, an elaboration is given on the case study: how it was conducted and what the data analysis yielded. Lastly, a discussion is provided detailing the findings and limitations.

Background

Software Development

Software development as a discipline follows different process models, depending on the area of application. All these software development life cycle models are structured in phases, representing the different tasks which are carried out in order to create a software product (Shylesh 2017). The potential influence of generative AI and machine learning (ML) on these life cycle phases differs. Over the past years, different possibilities of application have evolved. One example is the concept of Software 2.0 (Karpathy, A. 2017). Contrary to Software 1.0, where developers explicitly wrote code to define behaviour, Software 2.0 uses neural networks to model the behaviour required (Sheng et al. 2020). The Software 2.0 process primarily involves two stages: (1) creating a data processing pipeline and (2) feeding the pipeline's output into a neural network, which then executes the designated task (Carbin 2019). While neural networks have traditionally been applied in areas like language translation and image recognition, the rising popularity of Software 2.0 allows for the replacement of numerous traditional algorithms that define complex systems with a single trained neural network (Sheng et al. 2020). Software 2.0 has not only revolutionized sectors such as financial fraud detection (Roy et al. 2018) and self-driving cars (Tian et al. 2018), but has also been utilized to transform systems that traditionally did not use neural networks, such as database indexes (Kraska et al. 2018).

With generative AI, an alternative approach of applying AI to the software development life cycle arose. It is now possible to use generative AI to generate source code when building a software product. Given these new possibilities brought by AI, the dynamic landscape of enterprise software development necessitates that software professionals possess skills beyond technical expertise (Anton et al. 2020). Scholars have documented the evolving nature of IT professionals' roles, encompassing changes in job categories, personal attributes, and skills essential for success (Calitz et al. 2010; Gallivan et al. 2004; Jan et al. 2023; Wingard and Farrugia 2021). Recognizing that IT operates within a social context, sustained competitive advantage through IT requires investment in the human capital of proficient employees. In this context and by considering the insistent evolution of technology and its anticipated future prominence, it becomes essential to investigate the impact on skills and the future of work for software developers (Havstorm and Karlsson

2023).

Staying informed about developments in this field is vital for professionals, and one mechanism guiding the necessary knowledge in lifelong software developer career development is the Software Engineering Body of Knowledge (SWEBOK) (Agrawal 2010).

SWEBOK was formulated with five primary objectives. First, the aim is to foster a uniform perspective on software engineering in general, followed by delineating and clarifying its position in relation to other fields, such as computer science, project management, mathematics, and computer engineering. The third aim involves characterizing the contents of the software engineering discipline, ensuring easy access to relevant topics, and ultimately, serving as a foundation for curriculum development and individual certification (Bourque and Fairley 2014). SWEBOK's content and domain description are structured into 15 knowledge areas (KAs) grouped into four modules, as illustrated in Table 1. Software foundations encompass computing, mathematical, and engineering foundational KAs, while software development includes KAs covering software requirements, design, construction, and testing. The software management module comprises software maintenance, configuration and engineering management, and the engineering process KA. Lastly, software professional practice emphasizes engineering economics, software quality, engineering methods, and professional practices. The professional practice KA focuses on the knowledge, skills, and attitudes essential for software engineers, considering the impact of software products on social and personal life, personal well-being, and societal harmony. Software engineers are required to address distinctive engineering challenges, ultimately delivering software with recognized characteristics and reliability.

Software foundations	**Software management**	**Software development**	**Software professional practices**
Computing foundations	Software maintenance	Software requirements	Engineering economics
Mathematics foundations	Configuration management	Software design	Software quality
Engineering foundations	Engineering management	Software construction	Engineering methods
	Engineering process	Software testing	Professional practices

Table 1. Software Engineering Body of Knowledge (Bourque and Fairley 2014)

SWEBOK approaches KAs in a manner consistent with predominant schools of thought and typical industry breakdowns found in software engineering literature and standards. It does not assume specific application domains, business applications, management philosophies, development methods, etc. (Washizaki et al. 2023).

Software Development Productivity

Generally, productivity can be defined as the ratio of inputs to outputs, with a focus on increasing outputs (Koss and Lewis 1993). Efficiency differs from productivity as it pertains to reducing input while output remains constant. Given the multi-faceted nature of software development, determining software development productivity is complicated and has been extensively researched (Forsgren et al. 2021; Sadowski and Zimmermann 2019; Storey et al. 2022; Ziegler et al. 2022). A further distinction can be made between how software developers and their managers define productivity (Storey et al. 2022). From a managerial standpoint performance and the quality of certain outcomes typically relate to productivity, whereas developers view their daily tasks, such as writing code and working on pull requests, as being productive (Storey et al. 2022).

The SPACE framework provides a rationale for measuring developer productivity (Forsgren et al. 2021). It entails five dimensions, of which each describes one aspect that encapsulates software development productivity as a whole. These are outlined in Table 2:

Criteria	**Rationale**
Satisfaction and well-being	Fulfillment and happiness
Performance	Process outcome
Activity	Count of completed actions
Communication and collaboration	How people work together
Efficiency and flow	Minimal delays in work

Table 2. SPACE Framework (Forsgren et al. 2021)

The SPACE framework was developed collaboratively by GitHub, Microsoft and the University of Victoria (Gralha 2022). The measures themselves also might be difficult to determine; the creators of this framework proposed that it should not be viewed on its own but rather as relative across at least three dimensions (Alwell 2021; Forsgren et al. 2021). It can be applied to individuals and teams as well as systems (Forsgren et al. 2021). A full application is laid out in the section on the case study; however, in elaboration, when employees feel satisfied and happy with their work, it has been shown to correlate with an increase in productivity (Alwell 2021). How well goals are met depends on performance. This study is mindful in noting, for example, that large quantities of code do not automatically mean quality code, and quality code itself does not necessarily meet goals (Forsgren 2021). Developer activity intuitively relates to topics such as continuous integration and continuous delivery (CI/CD), the number of commits and lines of code. Certainly, this is valuable to consider, yet should never be a sole metric (Forsgren 2021; Forsgren et al. 2021). Clear communication is cardinal to productiveness; thus, by considering how developers collaborate, one would be able to gauge the quality of knowledge transfers, documentation discoverability as well as the thoroughness of code reviews (Forsgren 2021). Lastly, efficiency and flow relate to time — how many interruptions impede productivity and how timely code reviews are undertaken (Forsgren 2021).

An Automotive Case Study

Context

The objective of the case study is to measure the productivity of different teams in their customary working mode before the adoption of *Copilot* and after implementation. The utilization of *Copilot* was initiated by the case organization's IT management to increase developer productivity, increase standardization across development projects, and facilitate the adoption of new programming languages and frameworks. Preceding this case study, management made the decision to specifically introduce *Copilot*. It was selected based on an evaluation of *Copilot* compared to *Tabnine* and *AWS CodeWhisperer* where it was deemed that *Copilot* was more readily accessible and effective. The presented case study builds on this decision and therefore does not compare *Copilot* with other tools. As the possible effects on developer output have a potential impact on the planning of future IT initiatives, management commissioned a productivity measurement to learn more about the effects of *Copilot* on the company's software development process. The presented case study is one part of the overall productivity measurement approach. The organization has multiple IT Hubs across the world which all follow the same structures and processes. The findings of this case study are therefore used as an initial benchmark or proof of concept and will be applied to the other Hubs in the near future.

The *Copilot* version which is offered to the developers is "GitHub *Copilot for Business*". Providing this resource is the responsibility of a global toolchain department within the organization that focuses on developer tools. For all developers within the organization, the filter for suggestions matching public code is enabled for legal reasons. Consequently, no suggestions mentioning public code are presented to developers. Without this filter being enabled, the company could have been vulnerable to copyright infringement. The measurement outcome might be different due to this, however the organization still needs to operate within compliance guidelines and it is likely that most companies that employ similar generative AI technologies will have the same reasoning.

A key principle of this case study was that developers should work under conditions resembling their usual work routine as closely as possible. Therefore, the participating developers were allowed to use the IDE of their choice. Some developers use cloud-based IDEs like *AWS Cloud9*. The integration of *Copilot* into these cloud-based IDEs currently is not possible but they have additionally configured *Visual Studio Code* in order to use *Copilot*. Since one of the favorable characteristics of *Copilot* is the integration into an IDE, this setup was likely to affect the productivity gains for these developers. However, they still had access to *Copilot* as needed by simply copying code over to a working IDE.

The participating teams went through the same onboarding process provided to all other employees within the organization. This process is equal for all employees, regardless of their maturity level or project complexity. It had two relevant components regarding enablement. The first component was a basic compulsory onboarding session, conducted by the same department which is responsible for providing developer tools. During this session, the developers learned about compliance-related aspects of *Copilot* usage and received company access to *Copilot*. After this session they were technically enabled to use *Copilot*. Second, developers attended a *Copilot* prompting class presented by GitHub where they learned how to work with this technology. The content of this class was equal for all participants.

Approach

Five different teams were selected to participate out of the IT Hub's AI and data engineering departments. These teams were selected by management as they are code-intensive, have comparable sprints for before and after the adoption of *Copilot* and are representative of their department. Each team provides a specific solution to a business need in the organization and differs in terms of seniority level depending on the complexity of this need. Team 1 falls in the AI department and has three senior developers. They are responsible for building a business-to-business application requiring front-end and back-end development. Team 2 is a large data engineering team, consisting out of thirty developers that are predominately seniors. They focus on building a configurable digital twin data platform and is, therefore, very data-orientated. Team 3 is also a data engineering team that creates and populates data assets in the organization's data lake. They are fourteen developers with eight seniors. Teams 4 and 5 are AI teams; each with three senior and four junior developers respectively. Team 4 is responsible for building the organization's AI platform, and Team 5 for maintaining an advanced analytics product as well as developing corresponding front-end solutions.

In line with the SWEBOK KAs and the agile working model that the organization follows, including software requirements, design, construction, and testing (Bourque and Fairley 2014), the teams were measured from the beginning to end of a sprint. Sprints are part of the organization's agile working model and are fixed-length events where each sprint can be viewed as a short project (Schwaber and Sutherland 2020). Even though not all sprints were equal in terms of circumstances, great care was taken to identify sprints with as much similarity as possible, *ceteris paribus*, for each team.

Similar to other studies (Brynjolfsson et al. 2023; Wulf and Meierhofer 2023), this case study suggests that generative AI, specifically *Copilot*, will increase productivity. The productivity measurements identified as relevant for this study were throughput, cycle time, code quality, defects, and developer satisfaction. The measurements were identified by applying the SPACE framework (Forsgren et al. 2021). Further, these measurements align with other studies (Sadowski and Zimmermann 2019). The organization currently uses tools such as *JIRA*, *SonarQube*, *Seerene*, and *Confluence*. *JIRA* and *Confluence* are used daily by each team to track, document and plan their project work.[1] *SonarQube* is used to review code quality automatically.[2] *Seerene*, a powerful software process mining tool, works in combination with *JIRA* and *SonarQube* to measure throughput and code quality.[3] Thus, by leveraging these already in-use tools, the IT Hub was able to collect metrics for each team. This was collected both for a sprint where *Copilot* was not used and then again for a sprint where *Copilot* was actively used, allowing management to perform a comparative analysis

[1]Atlassian [website], https://www.atlassian.com/software/confluence/resources/guides/extend-functionality/confluence-jira#integrate-confluence-jira, (accessed 14 February 2024).
[2]SunarQube [website], https://www.sonarsource.com/ (accessed 14 February 2024).
[3]Seerene [website], https://www.seerene.com/ (accessed 14 February 2024).

by interpreting the measurements in the context of the SPACE framework. The teams were thus compared to themselves, not to each other. This approach allows the IT Hub to compare the performance of diverse teams over time, in an attempt to draw conclusions how *Copilot* affected their work.

It was relevant to this study that the *throughput* measured the volume of work completed by a team within a given time frame, such as a typical sprint. Two key data elements gauged productivity: First, the total number of story points completed per sprint, adjusted for the team size (i.e., divided by the number of developers). Story points are a unit of measurement for expressing an estimate of the overall effort required to fully implement a product backlog item or any other piece of work sourced from the work management tool *JIRA*. Second, the number of issues resolved, as analyzed by the code analytic platform, *Seerene*, was considered. This dual perspective helps to assess both the quantity and substantive contribution of the team's output. The *cycle time* quantifies the duration from the initiation to the completion of work items, specifically tracking the time elapsed from when a user story was marked as "in progress" to when it was classified as "done". This metric illuminated the team's efficiency at integrating and finalizing work, serving as a critical indicator of process agility and responsiveness. Cycle time data is typically extracted from *JIRA*. *Code quality* assessed the technical excellence of the code base, focusing on aspects like cognitive complexity, which reflected the ease with which code could be understood and modified. *SonarQube* is used to analyze the code quality in terms of code smells. High code quality is indicative of a maintainable, efficient, and robust software system. *Defects* refer to the number of issues or errors identified following the release of code into production. This metric was essential for evaluating the quality of the output and the effectiveness of the team's quality assurance processes. By tracking defects through *JIRA*, teams can monitor their impact on product stability and user satisfaction, guiding continuous improvement efforts and preventive strategies. Lastly, *developer satisfaction* describes how fulfilled developers feel in their work. Satisfaction measures the contentment and engagement of developers with their work and work environment. Data regarding this was obtained by asking the the teams directly over the course of the two sprints and their sentiment assessed (Forsgren 2021). Table 3 summarizes the measurements, together with the SPACE mapping and the relevant data collection tool used.

Measurement	SPACE mapping	Data collection tool
Throughput	Activity	JIRA, Seerene
Cycle time	Collaboration	JIRA, Seerene
Code quality	Performance	SonarQube, Seerene
Defects	Performance	JIRA, Seerene
Developer satisfaction	Satisfaction	Confluence (Survey)

Table 3. Measurements Application

Data Analysis and Findings

Throughput

Teams 1, 2 and 4 saw an increase in the amount of story points after adopting *Copilot*. As developers start using *Copilot* progressively, a reduction in story points might occur due to tasks being perceived as less complex. This was the case for teams 3 and 5 which alludes to an increase in their coding productivity. However, it might also be true that the increase in teams 1, 2 and 4's story points is due to them knowing that they will be using *Copilot* and planned their sprint accordingly. All teams saw a significant increase in the number of resolved issues, except for team 3 that solved half the number of their sprint without *Copilot*. From this, it seems developer activity was sustainable for teams 1, 2 and 4 as they were able to complete more tasks that were rated more difficult with the use of *Copilot*. Team 5 was more active in a less difficult sprint with the use of *Copilot* as they completed more issues than before adoption. This is interesting as it might suggest instances where developers feel more comfortable with using *Copilot* at times when they are dealing with easier tasks. Team 3's reduction in issues resolved during an easier sprint appears to indicate a decrease in activity. They have mentioned that due to unforeseen business problems, they spent most of the sprint on configuration and not development, but noted that *Copilot* aided in automating certain configuration tasks they previously did manually. This shows an increase in developer activity, albeit for work not directly related to their intially planned sprint.

Cycle time

Teams 2 and 4 remained constant whereas teams 1, 3 and 5 slightly improved. Often times when a new way of working is used, marked initial changes are expected. It is a valuable insight that there was not a massive change in cycle time as these teams were thus able to collaborate and communicate effectively before and after adoption. *Copilot* thus did not negatively impact the teams in terms of them working cohesively. Managers of software development teams should not have unrealistic expectations in terms of productivity gains, as the introduction of a new technology might not automatically lead to cycle time improvement.

Code quality

Teams 1, 2 and 3 showed significant improvement as their code-smells decreased. Teams 4 and 5 remained constant. Performance-wise it seems favourable that the code quality did not worsen after adopting *Copilot*. However, for this metric a risk exists in the collaboration between a human programmer and *Copilot*. While the dynamics of traditional pair programming among human developers are well-researched, there remains uncertainty surrounding the nature of collaboration between AI developers and humans (Wu and Koedinger 2023). Developers might assign too much trust to *Copilot* and simply accept proposed code without thinking about the quality, or the impact, which might lead to defects. Additionally, the performance of developers are impacted by their output quality, which relates to delivering on their tasks. This in turn may be heavily influenced by *Copilot*'s code completion functionality. *Copilot* might suggest a different, higher quality method of completing a task which the developer might adopt, ultimately increasing the team's code quality. The converse can also be true as previously noted. Insights from our data may be drawn that for this metric, managers should bear in mind that developer performance after adoption still depends on the developer's existing skills level. Experienced developers will know whether *Copliot* is aiding in code quality or not, and similarly, inexperienced developers could possibly learn from AI assistance (Ziegler et al. 2024).

Defects

Every team had less defects after adopting *Copilot*. This metric pertains to a developer's performance on solving production issues. Their code outcome dictates how well they performed. Interviews with an expert panel comprising some developers in the organization, highlighted that there is a risk that developers might accept code without properly validating the quality. The panel highlighted that *Copilot* will amplify the developers attributes. For example, it will turn a good developer into a very good one but might turn one who does not show due diligence in testing into a very poor developer. In this study, the adoption of *Copilot* did not lead to an increase in defects, however it is noteworthy to mention that defects usually occur during the course of a sprint, and is typically only solved in the future. If the adoption of *Copilot* caused defects, it would still be due to the fault of the developer.

Developer satisfaction

Asking developers directly is an efficient way to gauge their sentiment (Forsgren 2021). A Likert scale survey posed this question to the developers: "As a developer, my job satisfaction has increased or will increase due to the use of GitHub *Copilot*." Developers could select between "Strongly agree", "Agree", "Neither agree nor disagree", "Disagree" and "Strongly disagree". As part of software professional practices, it is crucial to consider the impact of software products on social and personal life, as well as their influence on personal well-being and societal harmony (Bourque and Fairley 2014). Therefore, developer satisfaction is not only important as a SPACE framework measurement for productivity but also because of the potential impact of generative AI, including *Copilot* on people (Ooi et al. 2023; Teubner et al. 2023). Similar to other studies (Kalliamvakou 2022), as regarding developer satisfaction, 80% of the developers in this study expressed the opinion that as a developer, their job satisfaction had increased or would increase due to the use of *Copilot*. As a first measurement this is very positive, however, the long-term use of generative technologies on people must still be investigated.

Conclusion

The use and integration of AI-assisted code completion technology into software development holds promise of increased productivity. However, this will depend on several conditions. This study showcased the intricacies involved with determining developer productivity when utilizing *Copilot*. Developers might need time for training as well as configuring their development environment, which could lead to an initial decrease in perceived productivity. The efficient utilization of *Copilot* will further be determined by the type of work done by the team, and how well the initial adoption of *Copilot* is managed within an organization. Our case study shows that a team's working environment needs to be right in order for *Copilot* to work; developers will reject a new technology if it impedes on their productivity. However, our case study fostered a positive working environment, with the general consensus being that *Copilot* is a tool that aids developers to be more productive. Additionally, the case study provided insights that it did not negatively affect developer activity and collaboration, since neither cycle time nor throughput declined. We recognize that developers would need to be willing to learn how to adopt *Copilot* and continue to use it for sustainable productivity increases. Our findings on code quality and defects emphasized that the *onus* is still very much on developers to perform professionally. Our study was limited by the relatively short time frame. A conclusion may be drawn that *Copilot* made developers feel more fulfilled in their work and serves as a modern way to beneficially approach developer problems, which in turn lead to an increase in productivity. Future research could include a long-term investigation and narrowing down on the effects that *Copilot* might have on junior and senior developers respectively. It would also be beneficial to investigate how different teams in the various international IT Hubs would fare. New or different measures of productivity would also yield meaningfully insights. Various alternatives exist to *Copilot* and comparative studies against these models should show interesting results. We recommend that future research should be conducted when developers do not feel they will be negatively impacted by the study, and alternatively to ours, possibly explore cases where developers have not had relevant training on efficient prompting.

REFERENCES

Agrawal, M. (2010). "Software 2.0–Emerging Competencies for Software Engineering Date: 7 August 2010 Organised by: Division II [Software] and Kolkata Chapter For details contact: Mr. Swarup Chakraborty, e-mail: csical@ gmail. com DCMC-10: Divisional Conference on Mobile Computing," in *National Conference on Mobile and Ad Hoc Networks,*

Alwell, K. (2021). "Measuring enterprise developer productivity," (available online at https://github.blog/2021-03-10-measuring-enterprise-developer-productivity/; accessed Feb. 12, 2024).

Anton, E., Behne, A., and Teuteberg, F. (2020). *The humans behind Artificial Intelligence–An operationalisation of AI competencies.*

Benbasat, I., Goldstein, D. K., and Mead, M. (1987). "The case research strategy in studies of information systems," *MIS Quarterly* (11:3), pp. 369–386.

Bourque, P. and Fairley, R. E. (2014). "SWEBOK v3. 0: Guide to the software engineering body of knowledge," *IEEE Computer Society* (), pp. 1–335.

Brynjolfsson, E., Li, D., and Raymond, L. R. (2023). *Generative AI at work*. Tech. rep. National Bureau of Economic Research.

Calitz, A., Greyling, J., and Cullen, M. (2010). "The South African ICT Work Environments and Graduate ICT Skill Requirements," in *International Business Conference (IBC), Victoria Falls, Zimbabwe,*

Carbin, M. (2019). "Overparameterization: A connection between software 1.0 and software 2.0," in *3rd Summit on Advances in Programming Languages (SNAPL 2019),* Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Oliveira Pinto, H. P. de, 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, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., 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., and Zaremba, W. (2021). *Evaluating Large Language Models Trained on Code*. arXiv: 2107.03374 [cs.LG].

Fahner, H. and Gerth-Ramos, J. (2023). "Revolutionizing development: Mercedes-Benz's journey with GitHub Copilot," Youtube. (Available online at https://www.youtube.com/watch?v=HdAqqNM1i9c; accessed Jan. 3, 2024).

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., and Zhou, M. (2020). *CodeBERT: A Pre-Trained Model for Programming and Natural Languages*. arXiv: 2002.08155 [cs.CL].

Forsgren, N. (2021). "Introducing Developer Velocity Lab to improve developers' work and well-being," Youtube. (Available online at https://www.youtube.com/watch?v=t7SXM7njKXw; accessed Feb. 2, 2024).

Forsgren, N., Storey, M.-A., Maddila, C., Zimmermann, T., Houck, B., and Butler, J. (2021). "The SPACE of developer productivity: There's more to it than you think," *Queue* (19:1), pp. 20–48.

Gallivan, M. J., Truex III, D. P., and Kvasny, L. (2004). "Changing patterns in IT skill sets 1988-2003: a content analysis of classified advertising," *ACM SIGMIS Database: the DATABASE for Advances in Information Systems* (35:3), pp. 64–87.

Gralha, C. (2022). "SPACE framework: 5 dimensions to measure developer productivity," (available online at https://blog.codacy.com/space-framework#:~:text=The%20SPACE%20framework%20is%20a,a%20holistic%20approach%20to%20productivity.; accessed Feb. 12, 2024).

Havstorm, T. and Karlsson, F. (2023). "Software developers reasoning behind adoption and use of software development methods–a systematic literature review," *International journal of information systems and project management* (11:2), pp. 47–78.

Jan, Z., Ahamed, F., Mayer, W., Patel, N., Grossmann, G., Stumptner, M., and Kuusk, A. (2023). "Artificial intelligence for industry 4.0: Systematic review of applications, challenges, and opportunities," *Expert Systems with Applications* (216), p. 119456.

Kalliamvakou, E. (2022). *Research: quantifying GitHub CoPilot's impact on developer productivity and happiness.*

Karpathy, A. (2017). "Software 2.0," (available online at https://karpathy.medium.com/software-2-0-a64152b37c35; accessed Jan. 14, 2024).

Koss, E. and Lewis, D. A. (1993). "Productivity or efficiency - Measuring what we really want," *National productivity review* (12), pp. 273–284.

Kraska, T., Beutel, A., Chi, E. H., Dean, J., and Polyzotis, N. (2018). "The case for learned index structures," in *Proceedings of the 2018 international conference on management of data,* pp. 489–504.

Moradi Dakhel, A., Majdinasab, V., Nikanjam, A., Khomh, F., Desmarais, M. C., and Jiang, Z. M. ((2023). "GitHub Copilot AI pair programmer: Asset or Liability?," *Journal of Systems and Software* (203) 2023.

Ooi, K.-B., Tan, G. W.-H., Al-Emran, M., Al-Sharafi, M. A., Capatina, A., Chakraborty, A., Dwivedi, Y. K., Huang, T.-L., Kar, A. K., Lee, V.-H., et al. (2023). "The potential of generative artificial intelligence across disciplines: Perspectives and future directions," *Journal of Computer Information Systems* (), pp. 1–32.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research* (21), pp. 5485–5551.

Roy, A., Sun, J., Mahoney, R., Alonzi, L., Adams, S., and Beling, P. (2018). "Deep learning detecting fraud in credit card transactions," in *2018 systems and information engineering design symposium (SIEDS),* IEEE, pp. 129–134.

Sadowski, C. and Zimmermann, T. (2019). *Rethinking productivity in software engineering,* Springer Nature.

Salva, R., Zhao, S., Kirschner, H., Androncik, A., and Schocke, D. (2023). "GitHub Copilot: the AI pair programmer for today and tomorrow," Youtube. (Available online at https://www.youtube.com/watch?v=AAT4zCfzsHI; accessed Jan. 3, 2024).

Schwaber, K. and Sutherland, J. (2020). "The Scrum Guide," (available online at https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf; accessed Apr. 18, 2024).

Sheng, Y., Vo, N., Wendt, J. B., Tata, S., and Najork, M. (2020). "Migrating a Privacy-Safe Information Extraction System to a Software 2.0 Design." in *CIDR,* Shylesh, S. (2017). "A study of software development life cycle process models," in *National Conference on Reinventing Opportunities in Management, IT, and Social Sciences,* pp. 534–541.

Storey, M.-A., Houck, B., and Zimmermann, T. (2022). "How developers and managers define and trade productivity for quality," in *Proceedings of the 15th International Conference on Cooperative and Human Aspects of Software Engineering,* pp. 26–35.

Teubner, T., Flath, C. M., Weinhardt, C., Aalst, W. van der, and Hinz, O. (2023). *Welcome to the era of ChatGPT et al.: The prospects of large language models*. 2023.

Tian, Y., Pei, K., Jana, S., and Ray, B. (2018). " DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars," in *Proceedings of the 40th International Conference on Software Engineering*. Pp. 303–314.

Washizaki, H., Sanchez-Segura, M.-I., Garbajosa, J., Tockey, S., and Nidiffer, K. E. (2023). "Envisioning software engineer training needs in the digital era through the SWEBOK V4 prism," in *2023 IEEE 35th International Conference on Software Engineering Education and Training (CSEE&T),* IEEE, pp. 122–126.

Wingard, J. and Farrugia, C. (2021). *The great skills gap: Optimizing talent for the future of work,* Stanford University Press.

Wu, T. and Koedinger, K. (2023). "Is AI the better programming partner? Human-human pair programming vs. human-AI pAIr programming," *arXiv preprint arXiv:2306.05153* () 2023.

Wulf, J. and Meierhofer, J. (2023). "Towards a taxonomy of large language model based business model transformation," *arXiv preprint arXiv:2311.0528* ().

Ziegler, A., Kalliamvakou, E., Li, X. A., Rice, A., Rifkin, D., Simister, S., Sittampalam, G., and Aftandilian, E. (2024). "Measuring GitHub Copilot's Impact on Productivity," *Communications of the ACM* (67:3), pp. 54–63.

Ziegler, A., Kalliamvakou, E., Li, X. A., Rice, A., Rifkin, D., Simister, S., Sittampalam, G., and Aftandilian, E. (2022). "Productivity assessment of neural code completion," in *Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming,* New York, NY, USA: ACM, 2022, pp. 21–29.