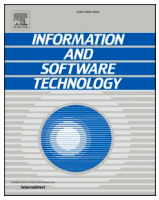
Information and Software Technology 183 (2025) 107751

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Copiloting the future: How generative AI transforms Software Engineering

Leonardo Banh [a,*], Florian Holldack [a], Gero Strobel [b]

[a] University Duisburg-Essen, Universitätsstr. 9, 45141 Essen, Germany
[b] University of Stuttgart, Keplerstraße 17, 70174 Stuttgart, Germany

ARTICLE INFO

Keywords:
Generative AI
Software Engineering
Information system development
Grounded Theory

ABSTRACT

Context: With rapid technological advancements, artificial intelligence (AI) has become integral to various sectors. Generative AI (GenAI) tools like ChatGPT or GitHub Copilot, with their unique content creation capabilities, pose transformative potential in Software Engineering by offering new ways to optimize software development processes. However, the integration into current processes also presents challenges that require a sociotechnical analysis to effectively realize GenAI's potential.

Objective: This study investigates how GenAI can be leveraged in the domain of Software Engineering, exploring its action potentials and challenges to help businesses and developers optimize the adoption of this technology in their workflows.

Method: We performed a qualitative study and collected data from expert interviews with eighteen professionals working in Software Engineering-related roles. Data analysis followed the principles of Grounded Theory to analyze how GenAI supports developers' goals, aligns with organizational practices, and facilitates integration into existing routines.

Results: The findings demonstrate several opportunities of GenAI in Software Engineering to increase productivity in development teams. However, several key barriers were also identified, that should be accounted for in successful integrations. We synthesize the results in a grounded conceptual framework for GenAI adoption in Software Engineering.

Conclusions: This study contributes to the discourse on GenAI in Software Engineering by providing a conceptual framework that aids in understanding the opportunities and challenges of GenAI. It offers practical guidelines for businesses and developers to enhance GenAI integration and lays the groundwork for future research on its impact in software development.

1. Introduction

With the latest advancements in digital technologies, we are seeing different parts of our lives being transformed, ranging from small-scale day-to-day interactions to large-scale organizational changes and new forms of working [1–3]. Artificial intelligence (AI) is one particular technology that has gained relevance in research and practice over the last decades, diffusing ubiquitously into our environment [4,5]. For instance, industrial quality assurance [6], home automation [7], or autonomous driving [8] make up intelligent systems that rely heavily on AI as the enabling technology at their base [9]. Therefore, even traditionally conservative industries now recognize AI's economic significance as a competitive differentiator [10,11]. The impact of AI is researched from multiple perspectives, including its capability to increase human productivity and its role as a collaborator in human-AI interaction [12–15]. Generative AI (GenAI) is an uprising type of AI that is capable of generating human-like content, such as text, images, or program code [16–18]. Well-known examples include the chatbot ChatGPT by OpenAI or GitHub Copilot, which assist users with creative, cumbersome, or complex tasks [19–21]. With its novel paradigm to enable a large user base through natural language prompting, GenAI poses potential to augment and automate processes that have been previously difficult to digitize. Consequently, a large user base is empowered to leverage GenAI for their individual tasks, underscoring the impact of GenAI and emphasizing the need for efficiency and competitiveness [5,22].

Software Engineering, a highly complex domain with well-trained professionals, faces increasing pressure to enhance productivity and quality while managing complexity and costs for the organization [23–25]. Traditional development processes, heavily reliant on human

* Corresponding author.
E-mail address: leonardo.banh@uni-due.de (L. Banh).

https://doi.org/10.1016/j.infsof.2025.107751
Received 6 September 2024; Received in revised form 26 March 2025; Accepted 3 April 2025
Available online 4 April 2025

expertise and manual coding, are being challenged by the availability of AI-assisted tools [20,23]. While developers can already benefit from task-aiding tools such as templates or auto-completion, the integration of GenAI into software development workflows presents new opportunities to optimize these processes due to GenAI's high contextuality and advanced reasoning capabilities [26,27]. Early studies suggest productivity gains through the application of GenAI in various programming tasks [28–30]. However, the integration of this technology also introduces challenges, such as concerns about code quality, security vulnerabilities, and potential skill erosion among developers, which may affect trust and adoption [23,31,32]. Especially with GenAI's natural language interaction paradigm, new patterns of work and collaboration arise that lead up to artificial counterparts as virtual coworkers [16,33,34]. As a result of humans utilizing GenAI for various tasks, traditional roles from human-machine interaction start to alter due to shifts in capabilities, responsibilities, and agency of AI systems [35,36]. Consequently, successful human-GenAI collaboration requires a profound analysis of GenAI embedded in its (professional) context to be leveraged by practitioners in complex organizations as well as researchers studying GenAI adoption.

Against this background, it is worthwhile investigating the potential and respective implications of introducing GenAI in the Software Engineering domain. In our study, we aim for a nuanced understanding of the action potentials and challenges of adopting GenAI as software developers. Hence, we ask the following research question:

RQ: *How does the integration of generative AI influence Software Engineering practices and which challenges are associated with its adoption?*

To address this question, we pursue a qualitative research approach, involving expert interviews with software developers, to develop a grounded conceptual framework on the adoption of GenAI in Software Engineering with its given potentials and implications. Grounded Theorizing following the Gioia methodology [37] allowed us to analyze the interview data methodologically and derive in-depth socio-technical insights of GenAI in its context of Software Engineering as well as identify its benefits and challenges for software developers. The results capture how GenAI facilitates developers' actions, aligns with their goals and values, and can be integrated into organizational routines and practices. Our study contributes to the ongoing discourse on the use of generative AI in Software Engineering [e.g., 23,38–41] by providing valuable insights for businesses and developers to effectively leverage GenAI in their workflows, and for the research community to further investigate potentials and challenges of GenAI for the domain of Software Engineering. The findings in the form of a conceptual framework will facilitate the development of guidelines and best practices for the effective adoption of GenAI in software development processes, offering a foundation for future research and practical implementation.

The remainder of this article is organized as follows. In Section 2, we illustrate the theoretical background of the paper. In Section 3, we outline our qualitative research approach and elaborate on the expert interviews as our primary data source. The results of our data analysis are presented in Section 4. Finally, we discuss the theoretical and practical implications of our findings in Section 5, together with limitations of our study, and conclude in Section 6.

2. Theoretical background

2.1. Generative AI

Recent advancements in the domain of AI have prominently enhanced its capabilities, thereby paving the way for many potential applications and introducing a new paradigm known as GenAI [16,42]. These advancements have endowed GenAI with the capacity to generate distinctive, realistic, and contextually congruent data that is nearly indistinguishable from human-generated content [19]. The multi-modal capabilities of various GenAI models enable these systems to create not only texts but also images, audio, and even more complex data types, such as programming code or molecules [19,20,43].

The fundamental technology that forms the core of recent GenAI systems is based on deep generative models (DGMs). In contrast to discriminative models, DGMs are designed to comprehend intricate data distributions, thereby enabling them to generate outputs that closely mirror real-world data [44]. The objective of DGM training is to learn high-dimensional probability distributions from finite training datasets and generate new, similar samples that approximate the underlying class of training data [45]. While discriminative models focus on the relationship between input features and output labels, generative models aim to learn the inherent data structure and generation processes [46]. Consequently, the objectives of DGMs diverge from those of traditional discriminative AI models in machine learning, as they prioritize the probabilistic generation of new data rather than determining decision boundaries (i.e., classification, regression, or clustering) for existing data [44,47]. Despite the existence of generative models for decades, such as Hidden Markov models and Bayesian networks designed for statistical problems involving time series or sequence [48], DGMs relying on neural networks have distinctly enhanced the quality of generated content (Fig. 1).

With novel capabilities and user-friendly interaction paradigms, such as natural language prompting for instruction and engagement, GenAI applications have opened the door to augment and automate processes that have traditionally been challenging to innovate [16,49]. These include cognitive demanding tasks (e.g., coding or developing user interfaces) as well as tasks that require human decision-making or empathic interactions (e.g., requirements analysis or team leadership). Consequently, a multitude of domains and industries, are investigating the potential integration of GenAI [22,50,51]. Initial research examines the application of generative AI in Software Engineering, with a particular focus on large language models (LLMs), which offer sophisticated reasoning capabilities [52,53]. These AI systems can assist in requirements analysis, generate code snippets, and provide insights for management, thereby streamlining processes that traditionally relied heavily on human expertise [20,54]. Additionally, they facilitate software testing and debugging by automatically generating test cases, identifying vulnerabilities, and suggesting optimizations [55,56]. As a result, generative AI holds the potential to drive disruptive innovation and assist software development teams in differentiating themselves in a competitive technology landscape [20,52].

2.2. GenAI in Software Engineering

GenAI has the potential to impact the field of Software Engineering from various perspectives. In particular, LLMs are often identified as a significant source of support for structured and knowledge-intensive software development tasks [23,53,55]. By automating labor-intensive tasks such as repetitive coding, testing, debugging or requirement traceability, GenAI is discussed to significantly enhance efficiency, freeing developers to focus on creative problem-solving and innovative design [20,57–59]. Moreover, these models have already demonstrated their advanced reasoning abilities and vast parameterized knowledge across a range of other application domains [54,60,61]. The reasoning abilities extend beyond natural language processing, encompassing the capacity to draw inferences and reach logical conclusions, based on the provided input alone [62,63]. This enables LLMs to handle more complex cognitive tasks associated with human intelligence, such as coding or medical diagnostic [64–66]. Consequently, GenAI offers the potential to assist developers throughout the entirety of the software development life cycle, from conceptualization and code generation to project analysis [55,56,67–70].

However, the widespread adoption of GenAI in Software Engineering faces several challenges. Primary concerns include those related to security [71–73], AI bias, and legal compliance issues [74,75]. Among these challenges, trust has emerged as a crucial area of research in order to gain a deeper understanding of GenAI in Software Engineering [73,

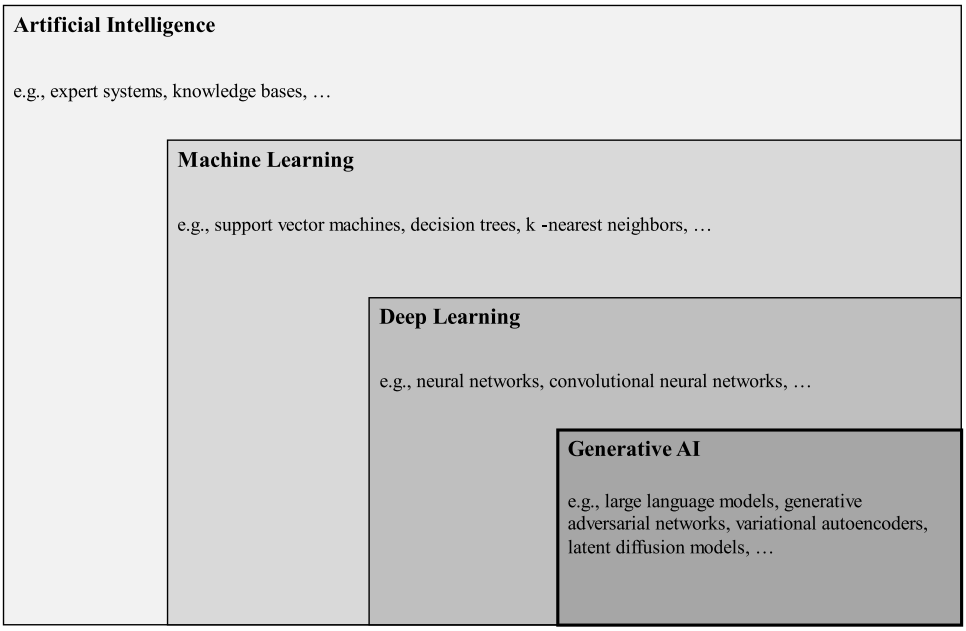


Fig. 1. Positioning of GenAI within the existing AI landscape [16].

76–78]. Previous studies have identified trust in programming tools as a key design requirement [20,73,79]. As AI systems grow more complex, developers' trust in GenAI-powered tools is shaped by factors such as the system's ability, benevolence, and integrity, as well as situational aspects such as the stakes of a given scenario or the complexity of the task [73]. Additionally, the explainability of GenAI plays a significant role in fostering trust, thereby influencing the adoption of GenAI in the field of Software Engineering [80]. In line with these findings, related research has suggested that traditional software testing paradigms may fall short when applied to the validation of AI tools due to their complexity [20, 41], thereby affecting trust in GenAI-based solutions.

Despite the challenges surrounding trust and explainability, GenAI introduces a novel interaction paradigm that enables users to engage more naturally with technology, paving the way for a new era of human-AI collaboration, such as AI pair programming [21,27,38,81]. This shift in how developers interact with AI tools has sparked extensive research into the productivity effects of GenAI on human workers [29,30,82,83]. The correctness and quality of generated code by GenAI tools such as GitHub Copilot plays an important factor in achieving enhanced productivity in software development processes [21,69,84]. However, different studies found varying degrees of correctness depending on the used programming language and task complexity [21,84–86]. While GenAI excels in well-defined and simpler tasks, its performance seems to decline as task complexity and input demands increase, thereby negatively impacting its adoption [87–89].

Related work by Russo [23] examined factors influencing the adoption of GenAI tools in Software Engineering. Using a mixed-method approach combining a questionnaire survey along with the creation of a validated theoretical framework, the author proposes the Human-AI Collaboration and Adaption Framework (HACAF) that provides insights regarding the individual drivers of GenAI adoption within existing software development workflows. Findings suggest that at this early stage of GenAI integration, the compatibility of GenAI tools with existing development framework is a primary driver for adoption whereas traditional technology acceptance factors such as perceived usefulness, social factors, and personal innovativeness seem less influential. While this primarily quantitative deductive study provides an initial understanding of the complexities of GenAI transformation within the field of Software Engineering, it lacks further in-depth investigations into the process changes as well as challenges resulting from developers' own experiences with GenAI.

The transformation of Software Engineering processes also acknowledges the changing role of developers. Recent studies suggest a shift toward developers spending more time reviewing AI-generated code while also revealing inefficiencies and time costs in interactions with AI [59,90]. On a broader scope, Sauvola et al. [52] propose different future scenarios for the evolving field of Software Engineering with AI, ranging from a traditional approach (with humans managing all roles) to a human-in-the-loop scenario (with autonomous AI entities and humans solely focusing on oversight-activities). Finally, related research takes an educational perspective on GenAI, including the consequences for teaching Software Engineering [91] and the learning effects with GenAI for novice programmers [92–95]. A lack of expertise might lead to non-optimal code solutions when AI outputs are not critically evaluated, calling for efforts in demystifying GenAI in general and making aware of challenges and limitations [21,95].

3. Method

Our study employs a qualitative research design centered on expert interviews to gain deep insights into the processes and tasks of software professionals and their opinions on GenAI [96]. We opted for an interview-based approach due to its capacity to generate rich, multi-perspective data grounded in experts' lived experiences [97,98]. To ensure a diverse yet relevant sample, we utilized purposive sampling to identify suitable experts with IT-related roles across various organizations. Our selection criteria focused not only on companies that purely develop software but also included employees that work in Software Engineering departments of other industries to adhere to a literal replication logic that enhances the generalizability of our findings across different organizational contexts [99]. Based on the collected interview data, we adopted the Grounded Theory method (GTM) to identify insights and emerging patterns that guide the construction of a conceptual framework [100,101].

3.1. Data collection

Our data collection spanned from August 2023 to January 2024, comprising semi-structured interviews with software developers from seventeen different companies headquartered in Europe. These

organizations operate across various sectors, including finance, energy, consulting, research, and software development (see Table 1). We developed an interview guide based on affordance-related questions[1] (i.e., our goal was to capture the respondents' expected or experienced immediate concrete outcomes of GenAI to achieve their individual goals in Software Engineering processes) [104,105,107] and inquiries about daily tasks, workflows, and personal views on GenAI and how it may evolve in the future of work (see Appendix A, Table A). For instance, we asked the participants about situations where to use or not use GenAI and their rationale behind or what has changed since GenAI tools became accessible. The semi-structured format allowed for flexibility, enabling us to adapt individually to the conversation flow and explore relevant topics in greater depth according to our research objectives [96,97,108].

All interviews were conducted virtually in German, subsequently recorded, transcribed, and translated into English. We encouraged participants to share openly their processes, opinions, and beliefs about generative AI in their day-to-day work and throughout the software development lifecycle. To ensure a comprehensive perspective, we selected participants from various IT-related positions [99], including developers, analysts, architects and managing roles, with experience levels in their current job tenure spanning from one to ten years to cover junior and senior roles. To draw the experts' connection to Software Engineering besides their IT-related profession, we also elicited their self-reported coding experience, i.e., the time a person has been programming for. A diverse range from one to fifteen years of coding experience highlights the participants' understanding and involvement in Software Engineering processes, thereby emphasizing how the interview data collected is applicable to GenAI in the domain of Software Engineering.

Table 1
Overview of interviewees.

ID	Industry	Role	Coding Experience	Job Tenure	Interview Length
1	Research	Research Assistant	10 years	4 years	21 min
2	Software Development	Lead Developer	10 years	5 years	36 min
3	Research	Research Assistant	3 years	4 years	20 min
4	Software Development	Senior Software Engineer iOS	13 years	5 years	19 min
5	Software Development	Head of Cloud & DevOps	12 years	5 years	26 min
6	Software Development	Solution Architect	10 years	10 years	24 min
7	IT Consulting	Head of Data & Intelligence	5 years	5 years	20 min
8	Organizational Development	Data Analyst	1 year	1 year	23 min
9	IT Consulting	Software Developer	3 years	5 years	18 min
10	Research	Senior Associate Researcher	15 years	6 years	31 min
11	Energy	Software Developer QM	5 years	5 years	25 min
12	Software Development	Business Analyst	5 years	1 year	22 min
13	IT Consulting	Consultant Enterprise Data & Analytics	8 years	6 years	16 min
14	Software Development	Software Engineer	15 years	9 years	28 min
15	Energy	Product Owner	5 years	2 years	20 min
16	Finance	Software Developer Mobile	10 years	8 years	23 min
17	Enterprise Software	Scrum Master	5 years	3 years	25 min
18	Software Development	Software Engineer Java	6 years	4 years	16 min

As suggested by the guidelines of GTM, our data collection process was closely intertwined with the data analysis phase, alternating between collection and initial analysis recursively until theoretical saturation was reached, i.e., additional data did not lead to novel emergent insights [109,110].

3.2. Data analysis

Given the nascent nature of generative AI, we adopted a Grounded Theory approach for analyzing the interview data [37,100]. GTM's flexibility is particularly suited for exploring phenomena with limited prior research [109]. As our research objective was to capture the experiences, opinions and the potential of GenAI to transform traditional Software Engineering processes in practice, the interview data guided us in developing socio-technical insights grounded in the specific emerging technology of GenAI while encompassing its social context in the domain of software development. This approach allowed us to identify the potential of GenAI for software developers, taking into account their unique characteristics, goals, and capabilities.

Our analysis followed the highly structured three-stage coding process of the Gioia methodology [37], ensuring qualitative rigor to code first-order concepts (similar to open coding by [101]) and derive second-order themes (similar to axial coding by [101]) that are distilled further into aggregate dimensions until theoretical saturation was achieved [37,111]. Adopting this method allowed us to develop an emerging theoretical model that guides our overall goal of synthesizing the potential of GenAI for Software Engineering. The qualitative data analysis tool MAXQDA supported us during the analysis process. To give an anchor example of the coding process, the raw code identified from the interview with expert 9 captured the following statement: "*The classic risk is that you might not devote as much time to simple topics yourself. I can also write an email to my colleagues beforehand. But if at some point you end up only using AIs for such small things, then the question is whether it's a bad thing to sort out simple things*" (Software Developer, 9). The emerging first-order concept on *questionable (code) reliability and accuracy* was summarized in the second-order theme of *reliability & dependency* that belongs to the aggregate dimension of *challenges*. We complied with grounded theorizing guidelines, continuously comparing emerging concepts and their linkages within the data [100]. After multiple iterations of coding by two independent researchers, theoretical saturation was reached with fourteen second-order themes in four aggregate dimensions. Fig. 2 illustrates our data structure, showcasing the progression from open first-order concepts to second-order themes, resulting in aggregate dimensions as third-order theoretical dimensions.

4. Results

After coding and analyzing the qualitative expert interviews, three main dimensions emerged, with *code quality improvement* and *reduced development time* both contributing to developers' overall goal to enhance productivity and *advanced reasoning capabilities* facilitating

[1] Affordance theory guided us as a theoretical lens [102,103] in creating a meaningful interview guide to capture the interaction between goal-oriented actors (i.e., software engineers) and a technological artifact (i.e., GenAI). This theory provides a unique lens through which researchers can examine the potential for actions that emerge from the relationship between human action and technology in organizational contexts [104,105]. Consequently, the interview guide aimed to shed light on how GenAI aligns with software developers' goals and values, and how it is integrated into existing work practices [106].

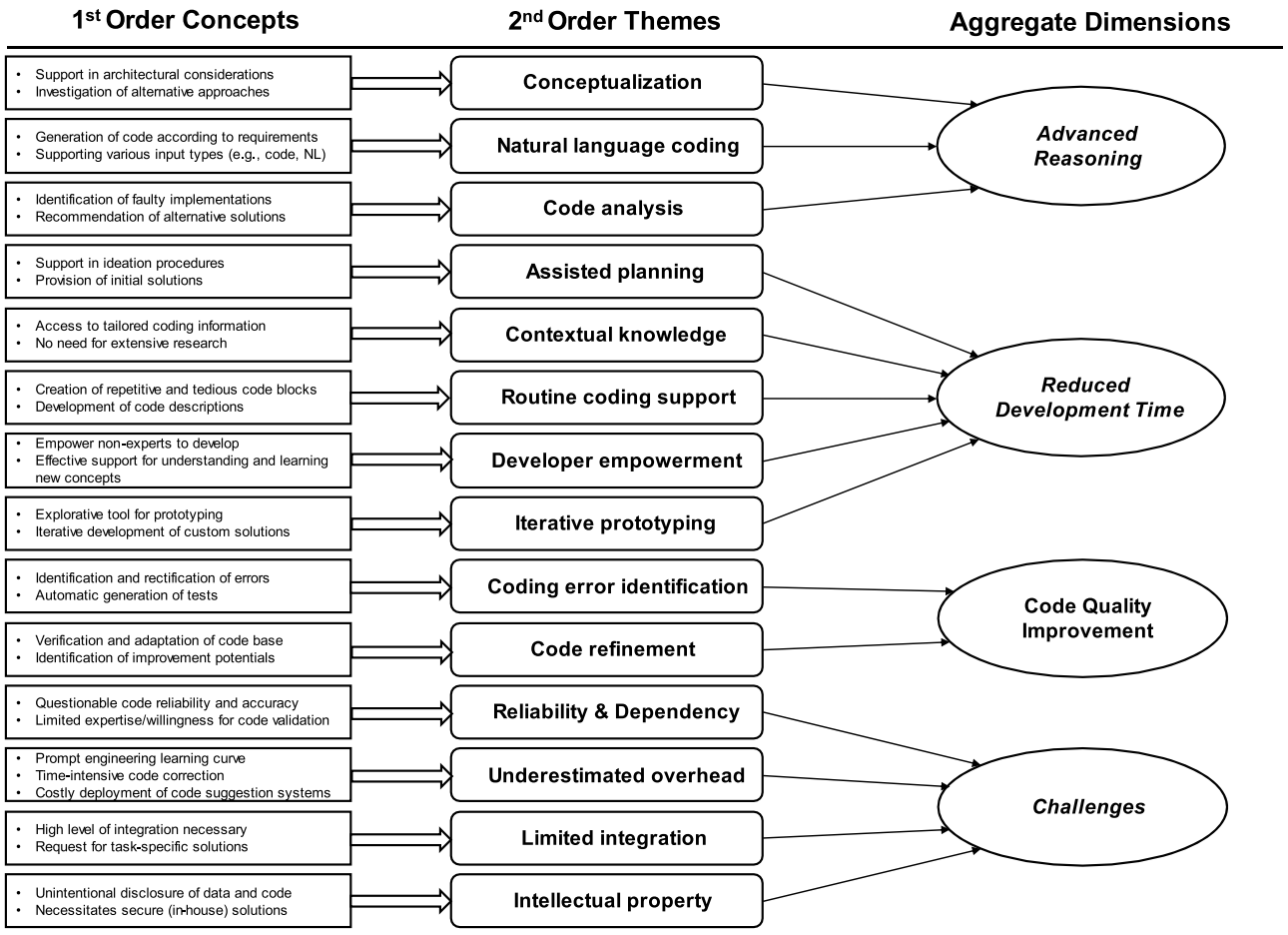


Fig. 2. Data structure.

GenAI's unique fit to the IT domain. In addition, we identified multiple *challenges* that affect the successful enactment of GenAI to support developers and should be accounted for.

4.1. Advanced reasoning capabilities

Conceptualization emerged as a major capability that supports software engineering tasks. Developers leverage LLMs' sophisticated reasoning capabilities and extensive knowledge in these early phases, thereby facilitating the generation and evaluation of concepts, ideas, or architectures: "*[…] describe how you want to do something and then you get quite solid feedback as to whether it makes sense or whether it doesn't, and how you could perhaps do it better*" (Software Engineer, 14). In this conceptualization stage, LLMs are not utilized for the direct generation of code; instead, they are employed for the evaluation of concepts in nascent states. This enables the identification of inadequate design proposals, thereby preventing further investment in unsuccessful directions. Moreover, LLMs are employed in an exploratory manner to investigate alternative approaches for solving problems: "*[…] get several solutions proposed and then decide on the basis of that […] what you can possibly derive from it*" (Consultant Enterprise Data & Analytics, 13). This significantly speeds up the corresponding development procedures: "*Because it simply speeds up the processes […] ChatGPT can deliver within two seconds*" (Product Owner, 15). As a result, the incorporation of GenAI in software development has the potential to significantly enhance both the speed and quality of the preliminary phases of software engineering.

Natural language coding is another fundamental capability of LLMs, which assists with core tasks of Software Engineering. It refers to the ability to not only comprehend and generate natural language but also to generate functional as well as contextually relevant programming code based on provided requirements: "*[…] now you have an additional option [GenAI], where you can also better define certain connections and the context or perhaps even the semantics and then get a result.*" (Solution Architect, 6). The findings from our interviews indicate that the most prevalent use case for language models in Software Engineering is the actual coding task: "*I use it quite a lot when it comes to the topic of coding*" (Research Assistant, 3). LLMs are not constrained by a specific input format or structure, thereby affording considerable flexibility and relieving cognitive load of developers: "*[…] because you don't have to stick to the form that the code generator needs, but you can simply use your own word, your own wording*" (Software Developer Mobile, 16). Thus, the inherent properties of LLMs facilitate the coding process by translating requirements into code or manipulating existing program code, thereby enhancing productivity: "*Yes, as I said, I think it is partly an increase in productivity, or it can definitely become one*" (Solution Architect, 6). Consequently, LLMs are employed to effectively outsource a substantial portion of major software engineering tasks.

Code analysis refers to the process of utilizing GenAI to examine existing code. The primary reported application is the identification of errors and the recommendation of corresponding optimizations: "*[…] I usually use it as, let's say, a tool for error identification. […] or […] to make the code a bit leaner […]*" (Data Analyst, 8). Accordingly, current LLMs are regarded as being capable of leveraging their comprehensive coding knowledge and reasoning abilities to examine existing code for potential shortcomings or areas for improvement. Furthermore, they can propose tailored solutions, thus making them invaluable tools for code analysis and quality control: "*[GenAI] solution that was also active with our tool […] to have it cross-checked […] I was very impressed by it*" (Software Developer QM, 11). In the event that an error cannot be handled by the

corresponding LLM, it can still assist the developers by facilitating the identification of it: "*If I can't find an error in the code, […] [GenAI] at least find clues to the error […] it worked quite well so far*" (Data Analyst, 8). As a result, the probability of errors or unoptimized code can be reduced, thereby enhancing the overall quality of the final code artifacts.

4.2. Reduced development time

Assisted planning is a useful use case for integrating GenAI in Software Engineering, in which the conceptualization capabilities of employed LLMs are leveraged to facilitate ideation and planning tasks: "*I mostly use it to gather initial ideas. Especially […] when […] something new […] is coming up, […]. Maybe also initial structuring of ideas […]*" (Software Developer, 9). In this context, GenAI offers a structured approach, conceptual guidance, and creative inspiration. Additionally, LLMs are employed to provide functional solutions as initial concepts for addressing problems: "*[…] I find it very pleasant to get ideas on how to solve a problem. […] You can also start and then just give ideas and ask, is there a better solution for this*" (Consultant Enterprise Data & Analytics, 13). Therefore, GenAI improves the preliminary phases of software development projects by speeding up the ideation and planning phases. Overall, this integration was described with notable time savings, which may subsequently enhance overall development productivity.

Contextual knowledge represents an important capability of GenAI, enabling the provision of tailored information and knowledge. This ability is of particular value in the context of Software Engineering as developers typically invest a substantial amount of time in extensive research activities, such as exploring state-of-the-art frameworks, or finding assistance in resolving errors: "*[…] I use it when I get stuck somewhere. And at that point I've googled […] maybe I also had to ask someone from my network or coworkers who also knows about development and my problems*" (Research Assistant, 1). LLMs have the potential to reduce the information retrieval activities, thereby enabling developers and engineers to obtain knowledge and information tailored to their specific task and problem situation: "*But that was of course much, much easier to type in two sentences than to go through fifteen stack overflow threads to see what the consensus is*" (Software Engineer, 14). The benefit of LLMs compared to conventional internet searches is that the data is presented in a concise and contextually relevant manner, which reduces the necessity for users to filter through relevant information from a vast repository of possible solutions in other use cases. As a result, GenAI tools are employed to augment and replace search engines and decrease the burden on other coworkers who would have been consulted for solutions prior to the integration of LLMs. This can contribute to considerable time and cost savings in software development teams: "*Because it simply speeds up the processes and otherwise, I would have to do a lot of things manually, […] ChatGPT can provide me with in two seconds*" (Product Owner, 15). In essence, the knowledge and reasoning capabilities of LLMs have the potential to result in significant efficiency gains, thereby promoting the overall productivity of software development processes.

Routine coding support emerged as a major factor influencing productivity, particularly in automating repetitive and tedious code snippets: "*Because a certain part is already tedious work, so somehow writing down classes where you already know in your head how it has to look. You just have to make the effort to do it*" (Software Developer Mobile, 16). While these tasks may not inherently be complex, they do result in an inefficient use of developers' time and capacity. Furthermore, the value of automating repetitive tasks is enhanced by the fact that portions of this code can often be repurposed in multiple projects: "*Yes, because coding is actually always very repetitive and code snippets can be used again and again*" (Consultant Enterprise Data & Analytics, 13). Although the potential for these repetitive coding tasks was characterized by low complexity and reusability, they are nevertheless a necessary component of the software development process and must be implemented in a fully functional manner. Moreover, other cumbersome routine tasks that are not directly related to coding could be enhanced by automation, including the creation of code documentation: "*[…] perhaps add descriptions so that the documentation effort is also reduced*" (Data Analyst, 8). Consequently, GenAI affords its integration into the workflow of software developers to minimize potential waste of time: "*I think that, especially when it comes to software development, generative AI products are very, very practical, especially when it comes to very simple, recurring code modules, methods […], you don't have to use the head or the brain of a developer […] many things […] that can be handled well with such tools without wasting time on them*" (Software Developer, 9). The outsourcing of repetitive and tedious coding tasks has a positive impact on productivity, as it allows developers to focus on unique, high-complexity tasks. Furthermore, it relieves the burden on senior developers and experts, who would otherwise be required to provide assistance to less experienced coworkers: "*[…] because I also do a bit of Android development now, and I'm far from an expert in that, […] can be, quite helpful, […] especially when it comes to such simpler stories […]*" (Software Engineer, 14).

Developer empowerment is as a particularly relevant feature by GenAI that effectively combines the aforementioned applications to facilitate individual productivity. Employed LLMs were perceived by the interviewed developers as knowledgeable discussion partners who provide valuable assistance to their workflows: "*And then to use it as […] a companion or as a control instance, […] so that I come to a solution faster and don't have to involve another person and […] take up their time*" (Data Analyst, 8). Accordingly, GenAI tools were seen as a partner that can provide support for various tasks, increasing the rigor and speed of developing a proper solution. This capability is beneficial when confronted with novel scenarios where one's expertise is limited. In these situations, GenAI can provide a first starting point, offering proper guidance: "*when I start a new topic now, I do it together with ChatGPT, because I have found that it is very helpful to get off to a good start*" (Lead Developer, 2). Moreover, GenAI facilitates the explanation of existing code and the generation of tailored responses to user inquiries about specific code components: "*[…] some of the Python code wasn't entirely comprehensible to me. […] That's why I simply entered the code and told [GenAI] to please explain or simplify it to me. […] I think it can be very helpful, especially if you're a junior*" (Lead Developer, 2). Sophisticated LLMs provide inexperienced users with the ability to effectively learn and understand existing code in various programming languages and contexts. Consequently, GenAI empowers users with limited expert knowledge in a specific field, enabling them to address more complex challenges that they would otherwise be unable to solve independently. This results in greater efficiency and effectiveness in problem-solving, which in turn increases development productivity of software engineers. Finally, the implementation of GenAI in Software Engineering empowers workers by facilitating quick verification of solutions, thereby reducing the perceived uncertainty of developers and enhancing the probability of implementing precise code: "*[…] I'm maybe looking for confirmation of what I have in mind, whether my solution actually […] makes sense. […] if an AI suggests the same thing, then the probability that you're roughly right […] should be rather high*" (Lead Developer, 2).

Iterative prototyping emerged as another valued application of GenAI in Software Engineering. The interaction with GenAI typically involves a trial and error approach, whereby developers prompt the language model multiple times, bearing in mind that not all of the responses are suitable for use: "*I would say that the process is characterized more by trial and error, because […] it is simply much easier to get more mass […]*" (Head of Data & Intelligence, 7). Accordingly, LLMs are employed as exploratory tools to identify potentially viable solutions and iteratively maturing the prototype state. This is made possible by the fast and simple acquisition of code suggestions and relevant data from currently available GenAI models: "*[GenAI] generates code really quickly.*" (Head of Data & Intelligence, 7). However, the solutions provided by GenAI tools do not always meet the developers' expectations or the project's functional requirements: "*I can send the same prompt three times to generate a code, and I get a different code each time. […] Whether the code runs the first*

time or not is a moot point, you can still develop it yourself" (Software Developer QM, 11). Therefore, it is not anticipated that a fully functioning code base will be produced in the initial iteration of prompting an LLM. Nonetheless, LLMs are capable of also modifying existing code based on further natural language instructions and descriptions by the software engineers. Thus, GenAI in Software Engineering is typically employed in an iterative approach instead of a one-shot approach, with the overarching objective of achieving a functional solution: "*Yes, because I can quickly find a suitable solution and get a quick suggestion […] but can first take a solution that is available and then think about […] how do I adapt the whole thing so that it is best for my problem*" (Consultant Enterprise Data & Analytics, 13).

4.3. Code quality improvement

Coding error identification is a fundamental ability during software engineering processes, necessary to develop high quality programs and enforce defined quality standards. The analytical capabilities of current LLMs, enabled through their inherent knowledge and reasoning capabilities, are already leveraged and valued by practitioners: "*[…] I can recognize patterns, or recognize errors in patterns that I don't recognize at first glance, […] or simply a syntax error and ChatGPT is not so bad at finding syntax errors*" (Data Analyst, 8). Furthermore, LLMs are highly effective at fixing identified errors, making the resolution of potential problems simple and timesaving: "*AI would help me a lot in fixing errors faster than I could on my own*" (Software Engineer Java, 18). Another option to integrate GenAI for error identification is the automatic generation of context-specific test cases: "*[…] generates test cases at the push of a button, because that's always one of the things that developers don't like to do, but it's very important […]*" (Lead Developer, 2). The generation of test cases is typically regarded as a routine task that does not fall within the scope of the high-complexity core responsibilities of a software engineer. Thus, LLMs can be deployed to generate useful and accurate tests, enabling developers to invest their time and mental effort in core tasks, thereby increasing productivity while ensuring the fulfilment of defined quality standards.

Code refinement builds upon the identification of errors and generation of test cases by leveraging GenAI's advanced reasoning capabilities to scan the code for other weaknesses and suggesting potential improvements, such as ways to enhance the readability or efficiency of specific code segments: "*Especially for verification. So, you can also use generative AI to read through code and see if it's comparable, […] make it leaner and more efficient*" (Software Developer Mobile, 16). Therefore, the utilization of LLMs enables the verification and adaptation of written code, which may ultimately enhance the overall quality of the code. Furthermore, the integration of GenAI as a second instance allows for a different perspective to be taken on the developed artefact, thereby facilitating the identification of optimization potentials that may have been overlooked by the corresponding developer: "*But of course, maybe some things wouldn't have been on your radar if you just didn't think about them at the time*" (Software Engineer, 14). Consequently, GenAI helps reducing coding errors caused by developers' lack of concentration, ultimately improving code quality. This further contributes to an improvement in productivity by enabling the developers to validate the provided software artifact in a timely manner.

4.4. Challenges

Reliability emerged as a significant concern regarding state-of-the-art LLMs due to their vulnerability to "hallucinations": "*[…] a lot of hallucination is also involved, i.e. false information is sold as true information*" (Head of Cloud & DevOps, 5). This phenomenon is characterized by responses that appear to be accurate and correct but are flawed by factual inaccuracies, a fact that is recognized by the majority of the interviewees: "*hallucinating is ultimately a problem at this point. Because, in the end, it only predicts what I want to hear or what is suitable for me*" (Lead Developer, 2). Accordingly, LLMs frequently fail to explicitly outline their limitations, instead generating false information or non-functional code: "*Because this AI can simply make things up or sometimes spit out things that don't even exist*" (Software Developer QM, 11). This can lead to frustration, prompting users to reconsider or limit their use of GenAI for coding tasks: "*[…] if I were to write an input that would then give me a code, I would have to put a lot more work into it to straighten it out. […] bad experiences with that […]*" (Data Analyst, 8). Consequently, this inherent uncertainty associated with the adoption and integration of DGM-based LLMs contributes to a reduction in the overall trustworthiness of GenAI. These trust issues are intensified by the high variability of generated outputs: "*I get a different code each time. And as soon as I realize that I lose a bit of confidence in the work that comes out of it*" (Software Developer QM, 11). The potential for hallucinations is especially pronounced in complex scenarios: "*[…] although I have to say that you always have to be extremely careful with the quality. […] A bigger problem occurs in very recent topics, such as the latest version of a framework with multiple deprecated methods or some new released package. There's also a lot of nonsense coming out of [GenAI]*" (Solution Architect, 6). Moreover, concerns were raised when leveraging current GenAI models for the generation of tests: "*But I think it's also a bit dangerous […] errors can creep in, […] if faulty tests are generated and you assume that everything is green […] that will slow down the team*" (Lead Developer, 2). Accordingly, the potential unreliability of current GenAI models can result in the generation of faulty test cases. This could further worsen the situation, as the developer may be confident that the system is functioning correctly, despite the presence of inherent test-related errors. This issue is aggravated by the tendency of humans to reduce critical thinking and to place absolute trust in the outputs of language models, thereby establishing a **dependency** on GenAI: "*Of course, sometimes there is almost the danger that you stop thinking for yourself a little. I think that might also be a problem in the future, that if AI is constantly available, people will tend to become lazy and stop thinking for themselves*" (Lead Developer, 2). Thus, GenAI has the potential to facilitate a cognitive decline of its users, resulting in significant implications for the future as individuals are reducing their own input and effort to verify information, investigate topics in depth, and showcase critical thinking: "*we have a loss of creativity here, but perhaps also a loss of education […] less independent thinking […] you have to find the answer and not somehow remember the answer yourself*" (Research Assistant, 1). It is therefore essential to identify an appropriate balance between harnessing the potential of GenAI for enhanced Software Engineering and avoiding an excessive reliance on these capabilities to offset the adverse consequences: "*So I think we need to find the right balance here, so that we get support, […] but don't become completely dependent on AI*" (Lead Developer, 2).

Underestimated overhead is a multifaceted challenge that emerges with the integration of GenAI in organizational contexts, partly resulting from the limited reliability of current LLMs. First, time-intensive corrections can be a direct consequence of hallucinations: "*But to a certain extent, there is always the issue of the effort required to ensure that there is no error or that it somehow increases*" (Senior Associate Researcher, 10). In light of these observations, the productivity gains associated with the use of GenAI in Software Engineering may be diminished by the need for manual corrections and optimizations to the AI-generated code: "*Therefore, you have certain productivity increases, but at the moment this is basically canceled out by your own quality control*" (Solution Architect, 6). Ultimately, this reduction in overall benefits underscores the need for solutions to address the issues of hallucination and inaccuracy. Second, the complexity of correctly prompting an LLM is another significant challenge, which shows as the difficulty in communicating objectives and expected results to the GenAI tool: "*Mainly, I still find it too much effort to communicate what I want*" (Research Assistant, 3). Consequently, it can require a significant investment of time to construct an effective prompt that will lead to the desired output: "*But there are also cases where you spend ages trying to cut the prompt in such a way that it somehow comes out the way you have it in your head […] it might have been easier to bring in*

a professional again" (Senior Associate Researcher, 10). Accordingly, the potential efficiency gains associated with the successful integration of GenAI models into the software development process may be also diminished by the need to invest more time than expected into prompting. This issue is worsened by the non-deterministic nature of DGMs, which can be observed with a single prompt yielding inconsistent outputs upon multiple iterations: *"I can send the same prompt three times to generate a code and I get a different code each time*" (Software Development QM, 11). Therefore, the actualization of GenAI's potential benefits can be a more time-intensive undertaking than expected. Third, concerns were raised regarding the potential costs associated with deploying GenAI models: "*So I think these are different values. How expensive will that be? Is this really more relief, or are these then costs, hidden costs?*" (Senior Software Engineer iOS, 4). These considerations are especially relevant in the context of pay-per-request pricing models, given that the variability of LLMs outputs often requires multiple calls to achieve the desired outcome: "*[…] then you are restricted there because, I think, you need a license, and then you pay per request or something like that*" (Research Assistant, 1). In combination with the need for code corrective measures, this challenge leads to concerns about the overall cost-benefit ratio for a sustainable GenAI adoption in large companies.

Limited integration refers to the expressed need for task-specific and highly integrated solutions. The problem currently lies in the limited practicality due to the lack of integration of various solutions into a single, well-suited program: "*I'd wish for more integration. Well, there are now a wide range of solutions, and I think it would be a real boost if they were integrated with each other […]*" (Head of Data & Intelligence, 7). Practitioners need solutions that can be integrated seamlessly into their software development procedures, combining the capabilities of several task-specific tools that exceed current integrated development environments (IDE) tools. However, the range of existing task-specific solutions is limited, which results in extra effort to develop well-suited solutions: "*There are no ready-made tools that I can use […] so the effort involved in actually creating your own tools or developing interfaces […] has actually increased*" (Senor Associate Researcher, 10). In addition, the integration of relevant context into prompts may further improve current GenAI models, thereby reducing hallucinations and increasing reliability: "*[GenAI] simply gets better […] because more data comes into play"* (Software Developer, 9). Especially for coding purposes, it is beneficial for LLMs to possess comprehensive knowledge of the entire project to generate code blocks and responses that are well-suited to the task and organizational knowledge: "*[…] AI tools that are integrated into the development environment and, so to speak, automatically knows the context. […] if it understands the complete context in order to then be able to ask questions about the code or to be able to suggest new code […] that would be great. Like a direct integration. […] You'd save on transaction costs*" (Lead Developer, 2). While the direct integration of GenAI into the coding environment simplifies the prompting procedure and can increase the correctness of generated output, it also raises data privacy concerns.

Intellectual property is identified as a major obstacle to the implementation of GenAI in business contexts: "*But the issue of data protection is still a big problem because it is not yet clear […] what exactly happens to the code. […] Just like the business logic that is implemented […] indirect data protection problems arise*" (Lead Developer, 2). The extent to which externally hosted language models in Software Engineering are protected from unintended disclosure of private information is unclear for the interviewed developers. This is particularly relevant for highly integrated systems that have access to large portions of code or entire projects. In this case, the code and data used as context in prompts are sent to external servers where they may be stored for some time or even used for training purposes: "*[…] these models also include context and save your entries, which means that the information is stored somewhere in the cloud and is not deleted for the time being*" (Research Assistant, 1). Furthermore, it is uncertain whether the utilization of a language model in a project would result in the unintended access to confidential files: "*[…] if you work with secret files, for example, access to the secret files is also theoretically possible*" (Lead Developer, 2). Consequently, experts, customers, and companies have expressed concerns regarding the utilization of GenAI in software development projects: "*We have also noticed recently that it is critical to assess the situation in our company, because there is currently no proper regulation in the area of data protection when you enter sensitive data as input. We are very careful about this now and I don't see it as something I would use daily*" (Data Analyst, 8). Accordingly, several companies and customers have prohibited using externally hosted GenAI models in their software development projects: "*[…] we have a very clear ban on that from the customer in my projects*" (Senior Software Engineer iOS, 4). Even when company policies do not restrict the use of LLMs for software development purposes, practitioners exercise caution regarding the integration of GenAI when sensitive data is involved: "*And if it involves sensitive data, personal data, company data, I would definitely not use it*" (Research Assistant, 1). Ultimately, the findings indicate a demand for sophisticated data protection mechanisms in the integration of GenAI for real-world development projects. Therefore, in-house on-premise solutions can be considered: "*If you could have it in-house, as a kind of hosted service from our AI experts, that would be phenomenal. I think that will become established*" (Software Developer Mobile, 16). These in-house solutions offer the advantage of complete data control, as the information is never transferred to external servers. This could facilitate the adoption of GenAI in various Software Engineering processes by enhancing its credibility and legal status internally. Nevertheless, the deployment and upkeep of these in-house solutions requires the involvement of AI and data protection experts, which may render an in-house GenAI solution inapplicable to small and medium-sized enterprises with limited resources.

5. Discussion

In light of the rapid advancements in generative AI and the increasing pressure to improve productivity and quality while managing complexity and costs in Software Engineering [23–25], our study examines the potentials and challenges of GenAI for software developers. Our findings demonstrate how GenAI can be employed in Software Engineering and outline its implications to enhance software developers' productivity by offering significant time savings and facilitating improvements in code quality. Furthermore, we identify key barriers to the adoption of GenAI in this context. The identified dimensions and their interconnections are depicted in a conceptual framework that offers empirical insights into the integration of GenAI within the domain of software engineering (see Fig. 3).

Our analysis highlighted the advanced reasoning capabilities of GenAI as the *enabling technology* capability for facilitating enhanced software development productivity. These GenAI capabilities can be applied throughout the entire software development lifecycle to support software engineers, from the initial conceptualization stage through the coding phase to the project analysis and optimization stage [67,112,113]. As LLMs demonstrate sophisticated reasoning abilities, their integration into software development workflows becomes increasingly compelling, particularly for the automation of substantial coding tasks [53,69,91]. Contemporary GenAI systems extend beyond mere code-assistance [55,56], offering additional support for assisting in architectural design decisions or other conceptualization tasks and improved code analysis, enabled by natural language interaction. This is accomplished by leveraging the large repository of trained coding knowledge and the capacity to transfer that knowledge to specific cases or problems, thereby making use of GenAI's advanced reasoning capabilities [23,42,55]. GenAI as an enabling technology helps streamlining workflows, reducing development time, and empowering developers to focus on more complex and creative aspects of their work [23,40,82]. Thus, the inherent reasoning and analysis abilities of current GenAI models allow them to address a wide range of tasks within the software development process, offering significant value to developers [20,55,69,

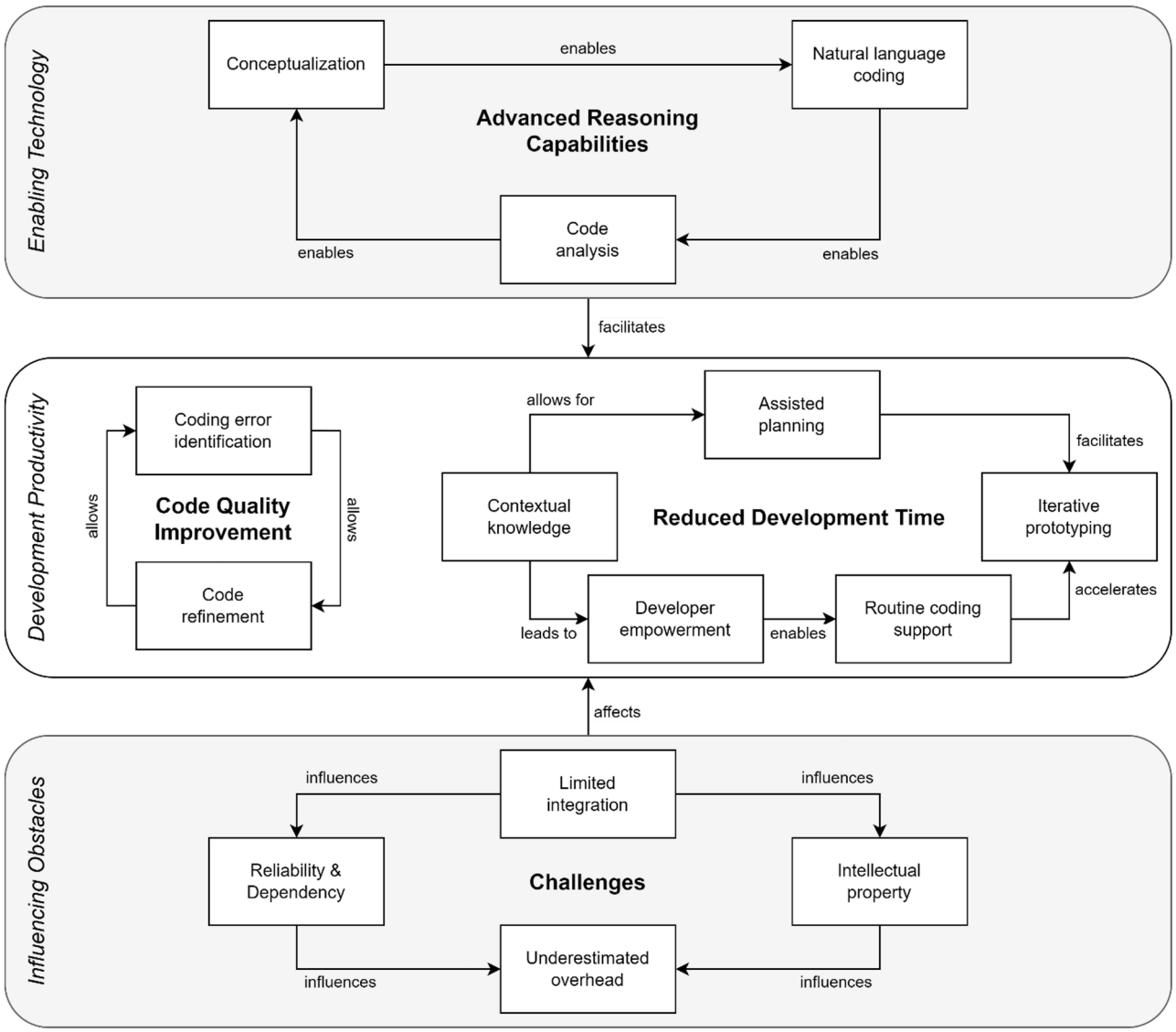


Fig. 3. Conceptual framework of generative AI integration in Software Engineering practices.

70].

This technological foundation facilitates the enhancement of Software Engineering *development productivity* by reducing the overall time and effort required, while simultaneously improving the quality of the resulting code output. While current IDEs are capable of identifying some types of errors including syntax faults, other types of errors, such as logical errors or security-related faults, may still be overlooked [56, 114]. The integration of LLMs and their contextual understanding extends this error identification procedure to allow for code refinement, thereby potentially increasing the overall code quality. Furthermore, contemporary GenAI models significantly reduce the time required to address Software Engineering tasks and problems, particularly by providing routine coding guidance and contextual knowledge. This aligns with the prevailing view in the literature, which highlights the potential for GenAI to enhance efficiency in Software Engineering [20, 57–59]. Consequently, the integration of GenAI into Software Engineering workflows may result in accelerated completion of a wide range of tasks, which facilitates its widespread adoption by automating routine coding, identifying errors, or refining code. In light of additional dimensions, such as iterative prototyping and developer empowerment, our findings suggest the emergence of a paradigm shift in perspective, whereby GenAI assistants are increasingly regarded as cooperative partners rather than mere tools. This transformation is particularly evident in the manner in which developers interact with GenAI models throughout the development process [21,27,38,81]. Instead of utilizing them as advanced code completion tools, developers are progressively engaging in an iterative discourse, wherein both parties contribute their respective capabilities: *"a partner in everyday working life […allowing] to go into a kind of discussion with the language models"* (Software Developer Mobile, 16). Additionally, the practitioners perceive the employed models as knowledgeable colleagues that enable them to tackle situations in which they are not experienced in This growing collaboration demonstrates the evolution of LLMs from simple tools to collaborative partners, a trend that aligns with the findings of existing human-computer-interaction literature [83,115]. As the level of integration increases, for instance with GitHub Copilot having access to entire projects, and as LLMs' reasoning capabilities grow, the autonomy of these systems is expanding [30,72,116]. Consequently, GenAI models become less dependent on user-supplied context data, as they are capable of retrieving the requisite information autonomously and even executing code to assess its functionality [58,85,116]. Particularly, when GenAI is deployed as a generative agent capable of operating in intricate environments with planning, acting, and reflecting on the outcomes [117–119], this shift in perception and utilization may be

significantly accelerated. Accordingly, the incorporation of state-of-the-art GenAI models into Software Engineering may contribute to the ongoing progression towards agentic information systems [35,120,121].

However, several *influencing obstacles* hinder the successful integration of GenAI into Software Engineering, including concerns about the reliability and dependency of LLMs for developers leading to underestimated overhead, concerns regarding data security and intellectual property, and challenges of limited integration of GenAI models into existing workflows [23,72,73,113]. Although high task complexity can exceed the capabilities of current GenAI models [87–89], an inherent uncertainty arises from the probabilistic nature of DGMs, which can result in erroneous responses and negatively impact accuracy or functionality [16,42]. This phenomenon gives rise to "hallucinations", which significantly impacts the reliability and trustworthiness of the model's outcome [16,42]. In our study, the majority of practitioners interviewed demonstrated a clear understanding of the aforementioned reliability issues. Nevertheless, there is a risk of overreliance and dependency on GenAI, particularly given the tendency of individuals to trust AI systems without critically evaluating their output in light of reducing their own effort [22,122,123]. Therefore, GenAI in Software Engineering has the potential to accelerate the phenomenon of humans becoming "cognitive misers" [124–126], despite knowing that there are issues regarding the reliability of GenAI. This tendency may be further intensified with the integration of more autonomous generative agents that are perceived as a knowledgeable and trustworthy colleague and could be explored separately from behavioral and psychological viewpoints.

These issues call for the implementation of comprehensive quality control measures, although such an approach may have an unfavorable impact on the development productivity [27,70,90,92]. Therefore, reliability represents a significant contributing factor to the emergence of an additional overhead. One strategy to mitigate these issues is to integrate GenAI models into existing development environments, thereby providing access to relevant data [86,116,127,128]. This type of integration facilitates the actualization of GenAI's potentials at a low level [23]; nevertheless, this degree of integration gives rise to concerns relating to data security and the potential disclosure of intellectual property [129,130]. Particularly with highly integrated and (semi-) autonomous agents such as GitHub Copilot that have the possibility of gaining access to a wide range of code and documentation, data security and privacy needs to be considered in businesses [30,72,116]. Consequently, there is a tension between the need for highly integrated and autonomous solutions and the imperative of data security.

A further concern is the fear of job displacement due to GenAI in various domains, particularly as autonomous agents approach the capabilities of artificial general intelligence [23,131]. Such concerns are shared by some practitioners, who anticipate that GenAI could reduce the number of developers needed by increasing individual efficiency: *"[…] certain jobs are no longer necessary or no longer require the amount of work. […] Five software developers right now […might be reduced] to two software developers in the future because they can work more efficiently with [GenAI]"* (Scrum Master, 30). Although the practitioners recognize the possibility of workforce reduction, they underscore that the human component is currently indispensable for sophisticated problem-solving and decision-making in software engineering. This conviction is reinforced by the current limitations of GenAI, which continues to require human supervision and manual input to function optimally, particularly in the context of high-stakes or intricate tasks [27,72,86]. Given these dynamics, an emerging consensus among practitioners suggests that the current path forward is not one of replacement, but rather one of cooperation [27,52,131]. By integrating GenAI into workflows as a collaborative partner, rather than a substitute, organizations can leverage the strengths of both human intelligence and AI to maximize productivity and quality.

In summary, our conceptual framework depicts how GenAI as an enabling technology through its advanced reasoning capabilities facilitates development productivity, primarily regarding improved code quality and reduced development time to yield results (see Fig. 3). Various Software Engineering tasks benefit from the integration of GenAI into daily work that allow for assisted planning and developer empowerment based on contextual understanding, ultimately leading to reduced development time. For instance, the support of routine coding procedures accelerates the time for iterative prototyping and is therefore perceived as increased development productivity. Nonetheless, influencing obstacles pose challenges to the successful adoption of GenAI in Software Engineering. Factors such as limited integration of GenAI into operations, concerns regarding reliability and dependency as well as intellectual property, and consequences of underestimated overhead all affect the integration of GenAI for software developers to pursue GenAI-supported increase in development productivity.

5.1. Implications

Our work not only highlights the transformative potential of GenAI in Software Engineering but also addresses the practical considerations that may influence its widespread implementation. Regarding the **theoretical implications** of our research, we contribute to the existing body of knowledge in human-AI interaction and AI-assisted software development [15,23,54,132] by providing an initial conceptual framework that offers empirical insights into the integration of GenAI within the domain of Software Engineering. Our framework of GenAI in Software Engineering facilitates further research efforts to study the sociotechnical implications, thereby deepening the current understanding of the field of GenAI adoption [133–135]. Second, our findings shed light on potential individual and organizational challenges associated with the integration of GenAI, which may have broader implications for other knowledge-intensive domains [1,5,14,136]. A particularly noteworthy field regarding the implications of GenAI adoption deals with its effect on productivity [28–30,82], resulting from a discussed task-technology fit and GenAI's cognitive offloading potential [12,36,137,138]. Therefore, this study offers valuable support to researchers that evaluate the antecedents and consequences of integrating GenAI into other knowledge-intensive domains [139,140].

Additionally, we present two interrelated **practical implications**. First, our conceptual framework (see Fig. 3) provides practitioners with guidance on effectively leveraging GenAI in Software Engineering. We illustrate potential applications of LLMs across a range of Software Engineering tasks and evaluate their individual benefits, thereby assisting practitioners in realizing the inherent potential of GenAI to foster development productivity and gain a competitive advantage. Consequently, this framework equips software development teams with the tools to identify and capitalize on the most promising opportunities afforded by generative AI in their specific contexts. This is relevant as current LLMs are already considered an indispensable tool for modern software developers. Second, we provide an initial overview of the existing challenges and potential pitfalls that must be considered when integrating GenAI into work procedures in professional contexts. Some challenges are intrinsic to the nature of DGMs, such as reliability issues, while others may arise from integrating this technology into human work processes. For instance, the potential loss of cognitive effort, as employees outsource tasks to AI systems, can be seen as an unintended consequence of GenAI adoption. It is, thus, imperative to acknowledge that the influence of generative AI on individuals can result in difficulties that call for considerations from a sociotechnical standpoint.

5.2. Limitations and outlook

Despite the contributions of our research, there are **limitations** that must be considered when interpreting the findings and creating avenues for **future work**. Since the results are derived from a qualitative study, they are inherently influenced by the sample of participants. However, the sampling process was designed to include a diverse group of experts

with different job roles, characteristics, and levels of expertise in generative AI to ensure a robust and comprehensive analysis. Additionally, it is important to note that the sample reflects the perspectives of these experts at a single point in time, which may evolve as the field of generative AI continues to develop. Long-term studies can investigate how fast GenAI adoption is happening across the industry and might provide opportunities to assess the maturity of current GenAI services and organizational readiness. As our resulting framework highlights the predominant theme of development productivity amongst our study participants, we encourage future studies to investigate additional effects (e.g., data security, developer identity, or AI bias) in more depth. Finally, our empirical study focused on the perceived individual opinions of developers regarding GenAI in their daily tasks. To measure an actual effect on the developer productivity, further experiment studies should be conducted that assess the variance and degree of support by GenAI for software development tasks. Besides task-related improvements, cognitive effects on the users could also be investigated.

6. Conclusion

In this research paper, we investigated how generative AI can be leveraged in the field of Software Engineering to enhance work and processes by improving productivity and quality while managing complexity and costs. By conducting an interview study with a diverse sample across multiple software engineering roles, we synthesize the potentials of GenAI to propose a grounded framework of opportunities and challenges in Software Engineering. We highlight the advanced reasoning capabilities of GenAI as an enabling factor and demonstrate how GenAI enhances development productivity by offering significant time-savings and facilitating improvements in code quality. Nonetheless, we also identify inherent challenges that call for adequate approaches to overcome the weakness of GenAI and foster technological innovation while respecting concerns such as data privacy and bias mitigation.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work, the authors used *DeepL, DeepL Write*, and *ChatGPT* to improve the readability and language of the manuscript. After using these services, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

CRediT authorship contribution statement

Leonardo Banh: Writing – review & editing, Writing – original draft, Visualization, Project administration, Methodology, Investigation, Data curation, Conceptualization. **Florian Holldack:** Writing – review & editing, Writing – original draft, Visualization, Methodology, Investigation, Data curation, Conceptualization. **Gero Strobel:** Writing – review & editing, Writing – original draft, Visualization, Supervision, Methodology, Investigation, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A

Table A
Interview questionnaire.

Introductory Questions	
1.1	Tell us about your software engineering experience (work occupation, job tenure, coding experience)
1.2	What do you understand by the term “Generative AI”?
Goal-oriented GenAI use	
2.1	What did you use Generative AI for the last time?
2.2	What else do you use the technology for (in general, not on the job)?
2.3	Which Generative AI systems do you use?
2.4	In what situations would you not use Generative AI?
2.5	What did Generative AI enable you to do, that was difficult or impossible before?
2.6	Where do you see application possibilities and potentials for yourself personally and for your company?
2.7	Do you know of any Generative AI projects in your company?
2.8	Follow-up question: Is generative AI already being used?
GenAI for Software Engineering	
3.1	What are the reasons why Generative AI does (not) fit your tasks as a software engineer?
3.2	Why is Generative AI (not) compatible with your work style?
3.3	Do you think the use of Generative AI relieves you in your work activities or does the use seem more like a burden?
3.4	What do you think about your future Generative AI usage? Why do you (not) want to use it?
Transformative Impact of GenAI	
4.1	What has changed through the use of Generative AI?
4.2	What did the process look like before and after using Generative AI?
4.3	What happened once you started to use Generative AI? (Benefits such as productivity or creativity)
4.4	What did it make it more difficult to do?
4.5	Were there things you expected to be able to do that were not in fact possible?
4.6	What risks do you see for the use of Generative AI?
Closing Questions	
5.1	What would you like to see in Generative AI technology for the future?
5.2	Would like to say anything that may has been forgotten to say?

Data availability

Data will be made available on request.

References

[1] R. Boyd, R.J. Holton, Technology, innovation, employment and power: does robotics and artificial intelligence really mean social transformation? J. Sociol. 54 (2018) 331–345, https://doi.org/10.1177/1440783317726591.

[2] G. Walsham, Are we making a better world with ICTs? Reflections on a future agenda for the IS field, J. Inf. Technol. 27 (2012) 87–93, https://doi.org/10.1057/jit.2012.4.

[3] C. Ebert, C.H.C. Duarte, Digital transformation, IEEE Softw. 35 (2018) 16–21, https://doi.org/10.1109/MS.2018.2801537.

[4] N. Berente, B. Gu, J. Recker, R. Santhanam, Special issue editor's comments: managing artificial intelligence, MIS Q. 45 (2021) 1433–1450.

[5] T. Eloundou, S. Manning, P. Mishkin, D. Rock, GPTs are GPTs: Labor market impact potential of LLMs, Science 384 (6702) (2024) 1306–1308, https://doi.org/10.1126/science.adj0998.

[6] R.S. Peres, X. Jia, J. Lee, K. Sun, A.W. Colombo, J. Barata, Industrial artificial intelligence in industry 4.0 - systematic review, challenges and outlook, IEEE Access 8 (2020) 220121–220139, https://doi.org/10.1109/ACCESS.2020.3042874.

[7] G. Evans, Solving home automation problems using artificial intelligence techniques, IEEE Trans. Consum. Electron. 37 (1991) 395–400, https://doi.org/10.1109/30.85542.

[8] K. Muhammad, A. Ullah, J. Lloret, J. Del Ser, V.H.C. de Albuquerque, Deep learning for safe autonomous driving: current challenges and future directions, IEEE Trans. Intell. Transp. Syst. 22 (2021) 4316–4336, https://doi.org/10.1109/TITS.2020.3032227.

[9] M. Negnevitsky, Artificial Intelligence: a Guide to Intelligent Systems, 3rd. ed., Addison-Wesley, Harlow, Munich, 2011.

[10] P.J. Ågerfalk, Artificial intelligence as digital agency, Eur. J. Inf. Syst. 29 (2020) 1–8, https://doi.org/10.1080/0960085X.2020.1721947.

[11] A. Agrawal, J.S. Gans, A. Goldfarb, Artificial intelligence: the ambiguous labor market impact of automating prediction, J. Econ. Perspect. 33 (2019) 31–50, https://doi.org/10.1257/jep.33.2.31.

[12] A. Fügener, J. Grahl, A. Gupta, W. Ketter, Cognitive challenges in human–artificial intelligence collaboration: investigating the path toward productive delegation, Inf. Syst. Res. 33 (2022) 678–696, https://doi.org/10.1287/isre.2021.1079.

[13] S. Amershi, D. Weld, M. Vorvoreanu, A. Fourney, B. Nushi, P. Collisson, J. Suh, S. Iqbal, P.N. Bennett, K. Inkpen, J. Teevan, R. Kikin-Gil, E. Horvitz, Guidelines for human-AI interaction, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, New York, NY, USA, Glasgow Scotland Uk, ACM, 2019, pp. 1–13.

[14] K. Sowa, A. Przegalinska, L. Ciechanowski, Cobots in knowledge work, J. Bus. Res. 125 (2021) 135–142, https://doi.org/10.1016/j.jbusres.2020.11.038.

[15] D. Wang, E. Churchill, P. Maes, X. Fan, B. Shneiderman, Y. Shi, Q. Wang, From human-human collaboration to human-AI collaboration. Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems, Honolulu HI USA, ACM, New York, NY, USA, 2020, pp. 1–6.

[16] L. Banh, G. Strobel, Generative artificial intelligence, Electron. Mark. 33 (2023) 63, https://doi.org/10.1007/s12525-023-00680-1.

[17] Y.K. Dwivedi, N. Kshetri, L. Hughes, E.L. Slade, A. Jeyaraj, A.K. Kar, A. M. Baabdullah, A. Koohang, V. Raghavan, M. Ahuja, H. Albanna, M. A. Albashrawi, A.S. Al-Busaidi, J. Balakrishnan, Y. Barlette, S. Basu, I. Bose, L. Brooks, D. Buhalis, L. Carter, S. Chowdhury, T. Crick, S.W. Cunningham, G. H. Davies, R.M. Davison, R. Dé, D. Dennehy, Y. Duan, R. Dubey, R. Dwivedi, J. S. Edwards, C. Flavián, R. Gauld, V. Grover, M.-C. Hu, M. Janssen, P. Jones, I. Junglas, S. Khorana, S. Kraus, K.R. Larsen, P. Latreille, S. Laumer, F.T. Malik, A. Mardani, M. Mariani, S. Mithas, E. Mogaji, J.H. Nord, S. O'Connor, F. Okumus, M. Pagani, N. Pandey, S. Papagiannidis, I.O. Pappas, N. Pathak, J. Pries-Heje, R. Raman, N.P. Rana, S.-V. Rehm, S. Ribeiro-Navarrete, A. Richter, F. Rowe, S. Sarker, B.C. Stahl, M.K. Tiwari, W. van der Aalst, V. Venkatesh, G. Viglia, M. Wade, P. Walton, J. Wirtz, R. Wright, "So what if ChatGPT wrote it?" Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy, Int. J. Inf. Manag. 71 (2023) 102642, https://doi.org/10.1016/j.ijinfomgt.2023.102642.

[18] M. Jovanovic, M. Campbell, Generative artificial intelligence: trends and prospects, Computer 55 (2022) 107–112, https://doi.org/10.1109/mc.2022.3192720.

[19] G. Strobel, L. Banh, F. Möller, T. Schoormann, Exploring generative artificial intelligence: a taxonomy and types, in: Proceedings of the 57th Hawaii International Conference on System Sciences, Hawaii, USA, 2024.

[20] C. Ebert, P. Louridas, Generative AI for Software Practitioners, IEEE Softw. 40 (2023) 30–38, https://doi.org/10.1109/MS.2023.3265877.

[21] A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M.C. Desmarais, Z. M. Jiang, GitHub copilot AI pair programmer: asset or liability? J. Syst. Softw. 203 (2023) 111734 https://doi.org/10.1016/j.jss.2023.111734.

[22] A. McAfee, D. Rock, E. Brynjolfsson. How to capitalize on generative AI: a guide to realizing its benefits while limiting its risks, Harvard Business Review, 2023.

[23] D. Russo, Navigating the complexity of generative AI adoption in Software Engineering, ACM Trans. Softw. Eng. Methodol. 33 (2024) 1–50, https://doi.org/10.1145/3652154.

[24] K. Petersen, Measuring and predicting software productivity: a systematic map and review, Inf. Softw. Technol. 53 (2011) 317–343, https://doi.org/10.1016/j.infsof.2010.12.001.

[25] C. de O. Melo, D.S. Cruzes, F. Kon, R. Conradi, Interpretative case studies on agile team productivity and management, Inf. Softw. Technol. 55 (2013) 412–427, https://doi.org/10.1016/j.infsof.2012.09.004.

[26] A. Woodruff, R. Shelby, P.G. Kelley, S. Rousso-Schindler, J. Smith-Loud, L. Wilcox, How knowledge workers think generative AI will (Not) transform their industries, in: Proceedings of the CHI Conference on Human Factors in Computing Systems, New York, NY, USA, Honolulu HI USA, ACM, 2024, pp. 1–26.

[27] D. Spinellis, Pair programming with generative AI, IEEE Softw. 41 (2024) 16–18, https://doi.org/10.1109/MS.2024.3363848.

[28] S. Noy, W. Zhang, Experimental evidence on the productivity effects of generative artificial intelligence, Science 381 (2023) 187–192, https://doi.org/10.1126/science.adh2586 (1979).

[29] S. Peng, E. Kalliamvakou, P. Cihon, M Demirer, The impact of AI on developer productivity: evidence from GitHub copilot, arXiv (2023), https://doi.org/10.48550/arXiv.2302.06590.

[30] A. Ziegler, E. Kalliamvakou, X.A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, E. Aftandilian, Measuring GitHub copilot's impact on productivity, Commun. ACM 67 (2024) 54–63, https://doi.org/10.1145/3633453.

[31] E. Brynjolfsson, D. Li, L. Raymond, Generative AI at Work, Q. J. Econ. 044 (2025), https://doi.org/10.1093/qje/qjae044.

[32] P. Hacker, A. Engel, M. Mauer, Regulating ChatGPT and other large generative AI models, in: Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency, New York, NY, USA, Chicago IL USA, ACM, 2023, pp. 1112–1123.

[33] W. Wang, G. Gao, R. Agarwal, Friend or foe? Teaming between artificial intelligence and workers with variation in experience, Manag. Sci. (2023), https://doi.org/10.1287/mnsc.2021.00588 mnsc.2021.00588.

[34] M.H. Jarrahi, Artificial intelligence and the future of work: human-AI symbiosis in organizational decision making, Bus. Horiz. 61 (2018) 577–586.

[35] A. Baird, L.M. Maruping, The next generation of research on IS use: a theoretical framework of delegation to and from agentic IS artifacts, MIS Q. 45 (2021) 315–341, https://doi.org/10.25300/MISQ/2021/15882.

[36] H. Maurya, A. Agrahari, A. Kumar, Human-AI collaboration: cognitive challenges in interacting with generative AI agents, in: Proceedings of the SIGHCI 2023, 2024.

[37] D.A. Gioia, K.G. Corley, A.L. Hamilton, Seeking qualitative rigor in inductive research, Organ. Res. Methods 16 (2013) 15–31, https://doi.org/10.1177/1094428112452151.

[38] S. Imai, Is GitHub copilot a substitute for human pair-programming?, in: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings, New York, NY, USA Pittsburgh Pennsylvania, ACM, 2022, pp. 319–321.

[39] I. Ozkaya, The next frontier in software development: AI-augmented software development processes, IEEE Softw. 40 (2023) 4–9, https://doi.org/10.1109/MS.2023.3278056.

[40] A. Rajbhoj, A. Somase, P. Kulkarni, V. Kulkarni, Accelerating software development using generative AI: ChatGPT case study, in: Proceedings of the 17th Innovations in Software Engineering Conference, Bangalore India, New York, NY, USA, ACM, 2024, pp. 1–11.

[41] A. Schmidt, Speeding up the engineering of interactive systems with generative AI, in: Proceedings of the Companion 2023 ACM SIGCHI Symposium on Engineering Interactive Computing Systems, Swansea United Kingdom, New York, NY, USA, ACM, 2023, pp. 7–8.

[42] S. Feuerriegel, J. Hartmann, C. Janiesch, P. Zschech, Generative AI, Bus. Inf. Syst. Eng. 66 (2024) 111–126, https://doi.org/10.1007/s12599-023-00834-7.

[43] B.L. Hie, V.R. Shanker, D. Xu, T.U.J. Bruun, P.A. Weidenbacher, S. Tang, W. Wu, J.E. Pak, P.S. Kim, Efficient evolution of human antibodies from general protein language models, Nat. Biotechnol. 42 (2024) 275–283, https://doi.org/10.1038/s41587-023-01763-2.

[44] J.M. Tomczak, Deep Generative Modeling, Springer International Publishing, Cham, 2022.

[45] L. Ruthotto, E. Haber, An introduction to deep generative modeling, GAMM Mitt. 44 (2021) e202100008, https://doi.org/10.1002/gamm.202100008.

[46] T. Jebara, Generative versus discriminative learning, in: T. Jebara (Ed.), Machine Learning, Springer US, Boston, MA, 2004, pp. 17–60.

[47] J. Weisz, M. Muller, J. He, S. Houde, Toward general design principles for generative AI applications, in: Proceedings of the Joint ACM IUI Workshops 2023, Sydney, Australia, 2023, pp. 130–144.

[48] H. GM, M.K. Gourisaria, M. Pandey, S.S. Rautaray, A comprehensive survey and analysis of generative models in machine learning, Comput. Sci. Rev. 38 (2020) 100285, https://doi.org/10.1016/j.cosrev.2020.100285.

[49] R. Schmidt, R. Alt, A. Zimmermann, Assistant platforms, Electron. Mark. 33 (2023), https://doi.org/10.1007/s12525-023-00671-2.

[50] A. Susarla, R. Gopal, J.B. Thatcher, S. Sarker, The Janus effect of generative AI: charting the path for responsible conduct of scholarly activities in information systems, Inf. Syst. Res. 34 (2023) 399–408, https://doi.org/10.1287/isre.2023.ed.v34.n2.

[51] P. Leonardi. Helping employees succeed with generative AI: how to manage performance when new technology brings constant and unpredictable change, Harvard Business Review, 2023.

[52] J. Sauvola, S. Tarkoma, M. Klemettinen, J. Riekki, D. Doermann, Future of software development with generative AI, Autom. Softw. Eng. 31 (2024) 26, https://doi.org/10.1007/s10515-024-00426-z.

[53] A. Moradi Dakhel, A. Nikanjam, F. Khomh, M.C. Desmarais, H. Washizaki, Generative AI for software development: a family of studies on code generation, in: A. Nguyen-Duc, P. Abrahamsson, F. Khomh (Eds.), Generative AI For Effective Software Development, Springer Nature Switzerland, Cham, 2024, pp. 151–172.

[54] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, J.M. Zhang, Large language models for Software Engineering: survey and open problems, in: Proceedings of the 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE), 2023, pp. 31–53.

[55] I. Ozkaya, Application of large language models to Software Engineering tasks: opportunities, risks, and implications, IEEE Softw. 40 (2023) 4–8, https://doi.org/10.1109/MS.2023.3248401.

[56] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, Q. Wang, Software testing with large language models: survey, landscape, and vision, IEEE Trans. Softw. Eng. 50 (2024) 911–936, https://doi.org/10.1109/TSE.2024.3368208.

[57] A. Aleti, Software testing of generative AI systems: challenges and opportunities, in: Proceedings of the 2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE), Melbourne, Australia, IEEE, 2023, pp. 4–14.

[58] S. Barke, M.B. James, N. Polikarpova, Grounded copilot: how programmers interact with code-generating models, Proc. ACM Program. Lang. 7 (2023) 85–111, https://doi.org/10.1145/3586030.

[59] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, I. Gazit, Taking flight with copilot, Queue 20 (2022) 35–57, https://doi.org/10.1145/3582083.

[60] A. Moradi Dakhel, A. Nikanjam, V. Majdinasab, F. Khomh, M.C. Desmarais, Effective test generation using pre-trained Large Language Models and mutation testing, Inf. Softw. Technol. 171 (2024) 107468, https://doi.org/10.1016/j.infsof.2024.107468.

[61] G. Strobel, L. Banh, What did the doctor say? Empowering patient comprehension with generative AI, in: Proceedings of the ECIS 2024, 2024.

[62] Y. Chang, X. Wang, J. Wang, Y. Wu, L. Yang, K. Zhu, H. Chen, X. Yi, C. Wang, Y. Wang, W. Ye, Y. Zhang, Y. Chang, P.S. Yu, Q. Yang, X. Xie, A survey on evaluation of large language models, ACM Trans. Intell. Syst. Technol. 15 (2024), https://doi.org/10.1145/3641289.

[63] M.M. Lucas, J. Yang, J.K. Pomeroy, C.C. Yang, Reasoning with large language models for medical question answering, J. Am. Med. Inform. Assoc. 31 (2024) 1964–1975, https://doi.org/10.1093/jamia/ocae131.

[64] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q.V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems, Curran Associates, Inc, 2022, pp. 24824–24837.

[65] M. Parmar, N. Patel, N. Varshney, M. Nakamura, M. Luo, S. Mashetty, A. Mitra, C. Baral, LogicBench: towards systematic evaluation of logical reasoning ability of large language models, in: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics, Bangkok, Thailand 1, Association for Computational Linguistics, 2024, pp. 13679–13707. Long Papers), ACL 2024August 11-16, 2024.

[66] B. Yang, S. Jiang, L. Xu, K. Liu, H. Li, G. Xing, H. Chen, X. Jiang, Z. Yan, DrHouse: an LLM-empowered diagnostic reasoning system through harnessing outcomes from sensor data and expert knowledge, in: Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 8, 2024, https://doi.org/10.1145/3699765.

[67] S. Shafiq, A. Mashkoor, C. Mayr-Dorn, A. Egyed, A literature review of using machine learning in software development life cycle stages, IEEE Access 9 (2021) 140896–140920, https://doi.org/10.1109/ACCESS.2021.3119746.

[68] S. Jalil, The transformative influence of large language models on software development, arXiv (2023), https://doi.org/10.48550/arXiv.2311.16429.

[69] R. Khojah, M. Mohamad, P. Leitner, F.G. de Oliveira Neto, Beyond Code generation: an observational study of ChatGPT usage in Software Engineering practice, Proc. ACM Softw. Eng. 1 (2024) 1819–1840, https://doi.org/10.1145/3660788.

[70] A. Bucaioni, H. Ekedahl, V. Helander, P.T. Nguyen, Programming with ChatGPT: how far can we go? Mach. Learn. Appl. 15 (2024) 100526 https://doi.org/10.1016/j.mlwa.2024.100526.

[71] M. Jaworski, D. Piotrkowski, Study of software developers' experience using the Github Copilot Tool in the software development process, arXiv (2023), https://doi.org/10.48550/arXiv.2301.04991.

[72] B. Zhang, P. Liang, X. Zhou, A. Ahmad, M. Waseem, Practices and challenges of using GitHub copilot: an empirical study, in: Proceedings of the 35th International Conference on Software Engineering and Knowledge Engineering, KSI Research Inc, 2023, pp. 124–129.

[73] R. Wang, R. Cheng, D. Ford, T. Zimmermann, Investigating and designing for trust in AI-powered code generation tools, in: Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency, New York, NY, USA, Rio de Janeiro Brazil, ACM, 2024, pp. 1475–1493.

[74] N.A. Ernst, G. Bavota, AI-driven development is here: should you worry? IEEE Softw. 39 (2022) 106–110, https://doi.org/10.1109/MS.2021.3133805.

[75] A. Nguyen-Duc, P. Abrahamsson, F. Khomh, Generative AI For Effective Software Development, Springer Nature Switzerland, Cham, 2024.

[76] W. Hasselbring, R. Reussner, Toward trustworthy software systems, Computer 39 (2006) 91–92.

[77] J.D. Lee, K.A. See, Trust in automation: designing for appropriate reliance, Hum. Factors 46 (2004) 50–80, https://doi.org/10.1518/hfes.46.1.50_30392.

[78] S. Lipner, The trustworthy computing security development lifecycle, in: Proceedings of the 20th Annual Computer Security Applications Conference, IEEE, 2004, pp. 2–13.

[79] E. Janhunen, T. Toivikko, K. Blomqvist, D. Siemon, Trust in digital human-AI team collaboration: a systematic review, in: Proceedings of the AMCIS 2024, 2024, p. 3.

[80] J. Sun, Q.V. Liao, M. Muller, M. Agarwal, S. Houde, K. Talamadupula, J.D. Weisz, Investigating explainability of generative AI for code through scenario-based design, in: Proceedings of the 27th International Conference on Intelligent User Interfaces, New York, NY, USA, Helsinki Finland, ACM, 2022, pp. 212–228.

[81] Q. Ma, T. Wu, K. Koedinger, Is AI the better programming partner? Human-human pair programming vs. Human-AI pAIr programming, in: Proceedings of the Workshop on Empowering Education with LLMs - the Next-Gen Interface and Content Generation 2023 Co-Located with 24th International Conference on Artificial Intelligence in Education (AIED 2023), Tokyo, Japan, 2023, pp. 64–77.

[82] M. Coutinho, L. Marques, A. Santos, M. Dahia, C. Franca, R. de Souza Santos, The role of generative AI in software development productivity: a pilot case study, arXiv (2024), https://doi.org/10.48550/arXiv.2406.00560.

[83] J. Chen, J. Zacharias, Design principles for collaborative generative AI systems in software development, in: M. Mandviwalla, M. Söllner, T. Tuunanen (Eds.), Design Science Research For a Resilient Future, Springer Nature Switzerland, Cham, 2024, pp. 341–354.

[84] A. Mastropaolo, L. Pascarella, E. Guglielmi, M. Ciniselli, S. Scalabrino, R. Oliveto, G. Bavota, On the robustness of code generation techniques: an empirical study on GitHub copilot, in: Proceedings of the 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, IEEE, 2023, pp. 2149–2160.

[85] N. Nguyen, S. Nadi, An empirical evaluation of GitHub copilot's code suggestions, in: Proceedings of the 19th International Conference on Mining Software Repositories, New York, NY, USA, Pittsburgh Pennsylvania, ACM, 2022, pp. 1–5.

[86] B. Yetistiren, I. Ozsoy, E. Tuzun, Assessing the quality of GitHub copilot's code generation, in: Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering, New York, NY, USA, Singapore Singapore, ACM, 2022, pp. 62–71.

[87] J. Cámara, J. Troya, L. Burgueño, A. Vallecillo, On the assessment of generative AI in modeling tasks: an experience report with ChatGPT and UML, Softw. Syst. Model. 22 (2023) 781–793, https://doi.org/10.1007/s10270-023-01105-5.

[88] H. Tian, W. Lu, T.O. Li, X. Tang, S.-C. Cheung, J. Klein, T.F. Bissyandé, Is ChatGPT the ultimate programming assistant – how far is it?, arXiv, (2023), doi:10.48550/arXiv.2304.11938.

[89] P. Vaithilingam, T. Zhang, E.L. Glassman, Expectation vs. experience: evaluating the usability of code generation tools powered by large language models, in: Proceedings of the CHI Conference on Human Factors in Computing Systems Extended Abstracts, New York, NY, USA, New Orleans LA USA, ACM, 2022, pp. 1–7.

[90] H. Mozannar, G. Bansal, A. Fourney, E. Horvitz, Reading between the lines: modeling user behavior and costs in AI-assisted programming, in: Proceedings of the CHI Conference on Human Factors in Computing Systems, New York, NY, USA, Honolulu HI USA, ACM, 2024, pp. 1–16.

[91] M. Wermelinger, Using GitHub copilot to solve simple programming problems, in: Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1, Toronto ON Canada, New York, NY, USA, ACM, 2023, pp. 172–178.

[92] R. Choudhuri, D. Liu, I. Steinmacher, M. Gerosa, A. Sarma, How far are we? The triumphs and trials of generative AI in learning Software Engineering, in: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, New York, NY, USA, Lisbon Portugal, ACM, 2024, pp. 1–13.

[93] A. Eckerdal, M. Thuné, A. Berglund, What does it take to learn 'programming thinking'?, in: Proceedings of the 2005 International Workshop on Computing Education Research - ICER '05, New York, New York, USA Seattle, WA, USA, ACM Press, 2005, pp. 135–142.

[94] M. Kazemitabaar, J. Chow, C.K.T. Ma, B.J. Ericson, D. Weintrop, T. Grossman, Studying the effect of AI code generators on supporting novice learners in introductory programming, in: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, New York, NY, USA, Hamburg Germany, ACM, 2023, pp. 1–23.

[95] O. Petrovska, L. Clift, F. Moller, R. Pearsall, Incorporating generative AI into software development education, in: Proceedings of the 8th Conference on Computing Education Practice, Durham United Kingdom, New York, NY, USA, ACM, 2024, pp. 37–40.

[96] C.B. Seaman, Qualitative methods in empirical studies of software engineering, IIEEE Trans. Softw. Eng. 25 (1999) 557–572, https://doi.org/10.1109/32.799955.

[97] M.D. Myers, M. Newman, The qualitative interview in IS research: examining the craft, Inf. Organ. 17 (2007) 2–26, https://doi.org/10.1016/j.infoandorg.2006.11.001.

[98] U. Schultze, M. Avital, Designing interviews to generate rich data for information systems research, Inf. Organ. 21 (2011) 1–16, https://doi.org/10.1016/j.infoandorg.2010.11.001.

[99] G. Paré, Investigating information systems with positivist case research, CAIS 13 (2004), https://doi.org/10.17705/1CAIS.01318.

[100] K. Charmaz, Constructing Grounded Theory, 2nd ed., SAGE, Washington DC, 2014. Los Angeles, London, New Delhi, Singapore.
[101] A. Strauss, J. Corbin, Basics of Qualitative Research: Techniques and Procedures For Developing Grounded Theory, 2nd ed., Sage Publications, Inc, Thousand Oaks, CA, US, 1998.
[102] C.S. Collins, C.M. Stockton, The central role of theory in qualitative research, Int. J. Qual. Methods 17 (2018), https://doi.org/10.1177/1609406918797475, 1609406918797475.
[103] L. Parker, Qualitative perspectives: through a methodological lens, Qual. Res. Account. Manag. 11 (2014) 13–28, https://doi.org/10.1108/QRAM-02-2014-0013.
[104] O. Volkoff, D.M. Strong, Critical realism and affordances: theorizing IT-associated organizational change processes, MIS Q. 37 (2013) 819–834, https://doi.org/10.25300/MISQ/2013/37.3.07.
[105] P. Leonardi, When flexible routines meet flexible technologies: affordance, constraint, and the imbrication of human and material agencies, MIS Q. 35 (2011) 147, https://doi.org/10.2307/23043493.
[106] C. Trocin, I.V. Hovland, P. Mikalef, C. Dremel, How Artificial Intelligence affords digital innovation: a cross-case analysis of Scandinavian companies, Technol. Forecast. Soc. Change 173 (2021) 121081, https://doi.org/10.1016/j.techfore.2021.121081.
[107] M.L. Markus, M. Silver, A foundation for the study of IT Effects: a new look at DeSanctis and Poole's concepts of structural features and spirit, JAIS 9 (2008) 609–632, https://doi.org/10.17705/1jais.00176.
[108] M.Q. Patton, Qualitative Research & Evaluation Methods: Integrating Theory and Practice, 4th ed., Sage, Washington DC, 2015. Los Angeles, London, New Delhi, Singapore.
[109] D.F. Birks, W. Fernandez, N. Levina, S. Nasirin, Grounded theory method in information systems research: its nature, diversity and opportunities, Eur. J. Inf. Syst. 22 (2013) 1–8, https://doi.org/10.1057/ejis.2012.48.
[110] K. Aldiabat, C.L. Le Navenec, Data saturation: the mysterious step in grounded theory method, TQR (2018), https://doi.org/10.46743/2160-3715/2018.2994.
[111] B. Saunders, J. Sim, T. Kingstone, S. Baker, J. Waterfield, B. Bartlam, H. Burroughs, C. Jinks, Saturation in qualitative research: exploring its conceptualization and operationalization, Qual. Quant. 52 (2018) 1893–1907, https://doi.org/10.1007/s11135-017-0574-8.
[112] T. Saravanan, S. Jha, G. Sabharwal, S. Narayan, Comparative analysis of software life cycle models, in: Proceedings of the 2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), 2020, pp. 906–909.
[113] L. Belzner, T. Gabor, M. Wirsing, Large language model assisted Software Engineering: prospects, challenges, and a case Study. Bridging the Gap Between AI and Reality, Springer Nature Switzerland, ChamCham, 2024, pp. 355–374.
[114] T. Kohn, B. Manaris, Tell me what's wrong: a Python IDE with error messages, in: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, New York, NY, USA, Association for Computing Machinery, 2020, pp. 1054–1060.
[115] M. Hamza, D. Siemon, M.A. Akbar, T. Rahman, Human-AI collaboration in Software Engineering: lessons learned from a hands-on workshop, in: Proceedings of the 7th ACM/IEEE International Workshop on Software-intensive Business, New York, NY, USA, Lisbon Portugal, ACM, 2024, pp. 7–14.
[116] Github, Github Copilot: The world's most widely adopted AI developer tool., 2024. https://github.com/features/copilot (accessed 20 August 2024).
[117] J.S. Park, J. O'Brien, C.J. Cai, M.R. Morris, P. Liang, M.S. Bernstein, Generative agents: interactive simulacra of human behavior, in: Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, New York, NY, USA, Association for Computing Machinery, 2023.
[118] A.R. Dennis, A. Lakhiwal, A. Sachdeva, AI agents as team members: effects on satisfaction, conflict, trustworthiness, and willingness to work with, J. Manag. Inf. Syst. 40 (2023) 307–337, https://doi.org/10.1080/07421222.2023.2196773.
[119] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W.X. Zhao, Z. Wei, J. Wen, A survey on large language model based autonomous agents, Front. Comput. Sci. 18 (2024) 186345, https://doi.org/10.1007/s11704-024-40231-1.
[120] A. Jakob, M. Schüll, P. Hofmann, N. Urbach, Teaming Up with intelligent agents — a work system perspective on the collaboration with intelligent agents, in: Proceedings of the ECIS 2024, 2024.
[121] J. Strunk, L. Banh, A. Nissen, G. Strobel, S. Smolnik, To delegate or not to delegate? Factors influencing human-agentic IS interaction, in: Proceedings of the ICIS 2024 2, 2024.
[122] U. León-Domínguez, Potential cognitive risks of generative transformer-based AI chatbots on higher order executive functions, Neuropsychology 38 (2024) 293–308, https://doi.org/10.1037/neu0000948.
[123] A. Klingbeil, C. Grützner, P. Schreck, Trust and reliance on AI — An experimental study on the extent and costs of overreliance on AI, Comput Human Behav 160 (2024) 108352, https://doi.org/10.1016/j.chb.2024.108352.
[124] K.E. Stanovich, SIX. The cognitive miser: ways to avoid thinking, in: K. E. Stanovich (Ed.), What Intelligence Tests Miss, Yale University Press, 2017, pp. 70–85.
[125] S.S. Sundar, J. Kim, Machine heuristic: when we trust computers more than humans with our personal information, in: Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, New York, NY, USA, Glasgow Scotland Uk, ACM, 2019, pp. 1–9.
[126] A. Fügener, J. Grahl, A. Gupta, W. Ketter, Will humans-in-the-loop become Borgs? Merits and pitfalls of working with AI, MIS Q. 45 (2021) 1527–1556, https://doi.org/10.25300/MISQ/2021/16553.
[127] H. Koziolek, S. Grüner, R. Hark, V. Ashiwal, S. Linsbauer, N. Eskandani, LLM-based and retrieval-augmented control code generation, in: Proceedings of the 1st International Workshop on Large Language Models for Code, New York, NY, USA, Association for Computing Machinery, 2024, pp. 22–29.
[128] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. Yih, T. Rocktäschel, S. Riedel, D. Kiela, Retrieval-augmented generation for knowledge-intensive NLP tasks. Advances in Neural Information Processing Systems, Curran Associates, Inc, 2020, pp. 9459–9474.
[129] D. Humphreys, A. Koay, D. Desmond, E. Mealy, AI hype as a cyber security risk: the moral responsibility of implementing generative AI in business, AI Ethics 4 (2024) 791–804, https://doi.org/10.1007/s43681-024-00443-4.
[130] R. Pasupuleti, R. Vadapalli, C. Mader, Cyber security issues and challenges related to generative AI and ChatGPT, in: Proceedings of the 2023 Tenth International Conference on Social Networks Analysis, Management and Security, IEEE, 2023, pp. 1–5.
[131] M.A. Kuhail, S.S. Mathew, A. Khalil, J. Berengueres, S.J.H. Shah, "Will I be replaced?" Assessing ChatGPT's effect on software development and programmer perceptions of AI tools, Sci. Comput. Program. 235 (2024) 103111, https://doi.org/10.1016/j.scico.2024.103111.
[132] Z. Chen, J. Chan, Large language model in creative work: the role of collaboration modality and user expertise, Manag. Sci. 70 (2024) 9101–9117, https://doi.org/10.1287/mnsc.2023.03014.
[133] L. Boussioux, J.N. Lane, M. Zhang, V. Jacimovic, K.R. Lakhani, The crowdless future? Generative AI and creative problem-solving, Organ. Sci. 35 (2024) 1589–1607, https://doi.org/10.1287/orsc.2023.18430.
[134] R. Sabherwal, V. Grover, The societal impacts of generative artificial intelligence: a balanced perspective, JAIS 25 (2024) 13–22, https://doi.org/10.17705/1jais.00860.
[135] S. Jarvenpaa, S. Klein, New frontiers in information systems theorizing: human-gAI collaboration, JAIS 25 (2024) 110–121, https://doi.org/10.17705/1jais.00868.
[136] M. Alavi, D.E. Leidner, R. Mousavi, Knowledge management perspective of generative artificial intelligence, JAIS 25 (2024) 1–12, https://doi.org/10.17705/1jais.00859.
[137] L. van Huy, H.T.T. Nguyen, T. Vo-Thanh, T. Nguyen, T.T.T. Dung, Generative AI, why, how, and outcomes: a user adoption study, THCI 16 (2024) 1–27, https://doi.org/10.17705/1thci.00198.
[138] S. Grinschgl, A.C. Neubauer, Supporting cognition with modern technology: distributed cognition today and in an AI-enhanced future, Front. Artif. Intell. 5 (2022) 908261, https://doi.org/10.3389/frai.2022.908261.
[139] S. Raisch, S. Krakowski, Artificial intelligence and management: the automation–augmentation paradox, Acad. Manag. Rev. 46 (2021) 192–210, https://doi.org/10.5465/amr.2018.0072.
[140] H. Benbya, F. Strich, T. Tamm, Navigating generative artificial intelligence promises and perils for knowledge and creative work, JAIS 25 (2024) 23–36, https://doi.org/10.17705/1jais.00861.