

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/375238625>

Generative Artificial Intelligence for Software Engineering – A Research Agenda

Preprint · October 2023

DOI: 10.2139/ssrn.4622517

CITATIONS

27

READS

4,576

15 authors, including:



Anh Nguyen Duc

University of South-Eastern Norway

168 PUBLICATIONS 2,286 CITATIONS

SEE PROFILE



Beatriz Cabrero-Daniel

University of Gothenburg

27 PUBLICATIONS 131 CITATIONS

SEE PROFILE



Adam Przybylek

Gdansk University of Technology

35 PUBLICATIONS 736 CITATIONS

SEE PROFILE



Dron Khanna

Free University of Bozen-Bolzano

25 PUBLICATIONS 253 CITATIONS

SEE PROFILE

Generative Artificial Intelligence for Software Engineering - A Research Agenda

Anh Nguyen-Duc^{a,b}, Beatriz Cabrero-Daniel^c, Adam Przybylek^d, Chetan Arora^e, Dron Khanna^f, Tomas Herda^g, Usman Rafiq^f, Jorge Melegati^f, Eduardo Guerra^f, Kai-Kristian Kemell^h, Mika Saariⁱ, Zheyang Zhangⁱ, Huy Le^{j,k}, Tho Quan^{j,k}, Pekka Abrahamssonⁱ

^a*University of South Eastern Norway, BøI Telemark, Norway3800,*

^b*Norwegian University of Science and Technology, Trondheim, Norway7012,*

^c*University of Gothenburg, Gothenburg, Sweden*

^d*Gdansk University of Technology, Gdansk, Poland*

^e*Monash University, Melbourne, Australia*

^f*Free University of Bozen-Bolzano, Bolzano, Italy*

^g*Austrian Post - Konzern IT, Vienna, Austria*

^h*University of Helsinki, Helsinki, Finland*

ⁱ*Tampere University, Tampere, Finland*

^j*Vietnam National University Ho Chi Minh City, Hochiminh City, Vietnam*

^k*Ho Chi Minh City University of Technology, Hochiminh City, Vietnam*

Abstract

Context Generative Artificial Intelligence (GenAI) tools have become increasingly prevalent in software development, offering assistance to various managerial and technical project activities. Notable examples of these tools include OpenAI’s ChatGPT, GitHub Copilot, and Amazon CodeWhisperer.

Objective. Although many recent publications have explored and evaluated the application of GenAI, a comprehensive understanding of the current development, applications, limitations, and open challenges remains unclear to many. Particularly, we do not have an overall picture of the current state of GenAI technology in practical software engineering usage scenarios.

Method. We conducted a literature review and focus groups for a duration of five months to develop a research agenda on GenAI for Software Engineering.

Results. We identified 78 open Research Questions (RQs) in 11 areas of Software Engineering. Our results show that it is possible to explore the adoption of GenAI in partial automation and support decision-making in all

software development activities. While the current literature is skewed toward software implementation, quality assurance and software maintenance, other areas, such as requirements engineering, software design, and software engineering education, would need further research attention. Common considerations when implementing GenAI include industry-level assessment, dependability and accuracy, data accessibility, transparency, and sustainability aspects associated with the technology.

Conclusions. GenAI is bringing significant changes to the field of software engineering. Nevertheless, the state of research on the topic still remains immature. We believe that this research agenda holds significance and practical value for informing both researchers and practitioners about current applications and guiding future research.

Keywords: Generative Artificial Intelligence, GenAI, ChatGPT, CoPilot, Software Engineering, Software Project, Software Development, Focus Group, Literature review, Structured review, Literature survey, Research Agenda, Research Roadmap

1. Introduction

In recent years, Generative Artificial Intelligence (GenAI) has attracted significant attention and interest from Software Engineering (SE) research and practice. GenAI tools such as GitHub Copilot¹ and ChatGPT² have been rapidly explored in various professional contexts given their remarkable performance in producing human-like content. The emerging adoption of these tools reintroduces a host of classic concerns about productivity and quality when adopting new technologies. Clearly, code generation and test case optimization are SE tasks that directly benefit from recent large language models (LLMs) [1, 2, 3, 4]. Beyond traditional research on applied Machine Learning (ML), GenAI tools introduced usability and accessibility, leveraging AI-generated content to a broader range of professionals, requiring less technical competence to work and integrate them into existing work environment. GenAI tools are also showing their potential in non-coding tasks, such as assisting requirements engineering, software processes and project management.

¹<https://github.com/features/copilot>

²<https://openai.com/chatgpt>

At the moment, GenAI is an active research areas with several challenges and open questions. To do well on specific tasks, LLMs require fine-tuning or training. Research has much focused so far on achieving reliable and scalable GenAI output for different SE tasks. GenAI models are inherently nondeterministic: the same prompt produces different answers on different inference executions [5]. Moreover, GenAI’s output can be very sensitive to the input parameters or settings [6, 7]. Hallucination is another common concern with AI-generated content, as they can be imaginary or fictional [8]. The promise of AI automation might be closer than ever in SE [9], when these open challenges are sufficiently addressed.

Research agenda is a popular type of work for guiding and organizing research efforts in certain research areas [10, 11, 12, 13, 14, 15]. It often includes a review of existing work, and a presentation of directions, visions and priorities, enabling researchers to make meaningful contributions to their respective fields and to address pressing challenges. This research agenda is driven by past and current work on GenAI for SE. Based on focus groups and a literature review, we portray the current research on GenAI in SE and present the research agenda with open challenges for future research. While we do not emphasize the comprehensiveness of the literature review due to the fast-moving nature of the topic, the focus groups present practical expectations of the future roles of GenAI in software development. The research agenda is organized into 11 areas of concern, which are: (1) Requirements Engineering, (2) Software Design, (3) Software Implementation, (4) Quality Assurance, (5) Software Maintenance and Evolution, (6) Software Processes and Tools, (7) Engineering Management, (8) Professional Competencies, (9) Software Engineering Education, (10) Macro Aspects and (11) Fundamental concerns of GenAI.

Two international events have provided a ground for this work. As a part of the first international workshop on AI-assisted Agile Software Development³, we ran our first focus group to explore the benefits and challenges of AI-assisted Agile software development. At the Requirements Engineering (RE) conference, we organized the second international workshop on Requirements Engineering for Software Startups and Emerging Technologies (RESET)⁴ with a special theme on GenAI and Requirements Engineering.

³<https://www.agilealliance.org/xp2023/ai-assisted-agile/>

⁴<https://resetworkshop.org/>

Consequently, the identified open challenges reflect not just the existing gaps in literature but also draw from the practical, value-driven insights of the co-authors.

The paper is structured as follows: we clarify the meaning of GenAI and what we know about GenAI in SE in Section 2, and we will present our research approach in Section 3. Section 4 presents our Research Agenda. Finally, Section 5 is Outlook and Conclusions.

2. Background

Application of AI/ML has a long history in SE research [16, 17, 18]. The use of GenAI specifically, however, is a more emerging topic. While the promise of GenAI has been acknowledged for some time, progress in the research area has been rapid. Though some papers have already explored the application of GPT-2 to code generation [19], for example, GenAI was not a prominent research area in SE until 2020. Following the recent improvements in the performance of these systems, especially the release of services such as GitHub Copilot and ChatGPT-3, research interest has now surged across disciplines, including SE. Numerous papers are currently available through self-archiving repositories such as arXiv⁵, and paperwithcode⁶. To prepare readers for further content in our research agenda, we present in this section relevant terms and definitions (Section 2.1), the historical development of GenAI (Section 2.2) and fundamentals on Large Language Models (Section 2.3)

2.1. Terminologies

Generative modelling is an AI technique that generates synthetic artifacts by analyzing training examples, learning their patterns and distribution, and then creating realistic facsimiles [20]. GenAI uses generative modelling and advances in deep learning (DL) to produce diverse content at scale by utilizing existing media such as text, graphics, audio, and video. The following terms are relevant to GenAI:

- *AI-Generated Content (AIGC)* is content created by AI algorithms without human intervention.

⁵<https://arxiv.org/>

⁶<https://paperswithcode.com/>

- *Fine-Tuning (FT)* updates the weights of a pre-trained model by training on supervised labels specific to the desired task [21].
- *Few shot* training happens when an AI model is given a few demonstrations of the task at inference time as conditioning [21].
- *Generative Pre-trained Transformer (GPT)*: a machine learning model that uses unsupervised and supervised learning techniques to understand and generate human-like language [22].
- *Natural Language Processing (NLP)* is a branch of AI that focuses on the interaction between computers and human language. NLP involves the development of algorithms and models that allow computers to understand, interpret, and generate human language.
- *Language model* is a statistical (AI) model trained to predict the next word in a sequence and applied for several NLP tasks, i.e., classification and generation [23].
- *Large language model (LLM)* is a language model with a substantially large number of weights and parameters and a complex training architecture that can perform a variety of NLP tasks, including generating and classifying text, conversationally answering questions [21]. While the concept *LLM* is currently widely used to describe a subset of GenAI models, especially out on the field and in the media, what exactly constitutes 'large' is unclear and the concept is in this regard vague. Nonetheless, given its widespread use, we utilize the concept in this paper as well, acknowledging this limitation.
- *Prompt* is an instruction or discussion topic a user provides for the AI model to respond to
- *Prompt Engineering* is the process of designing and refining prompts to instruct or query LLMs to achieve a desired outcome effectively.

2.2. History of GenAI

To better understand the nature of GenAI and its foundation, we present a brief history of AI in the last 80 years:

1. Early Beginnings (1950s-1980s): Since 1950s, computer scientists had already explored the idea of creating computer programs that could generate human-like responses in natural language. Since 1960s, expert systems gained popularity. These systems used knowledge representation and rule-based reasoning to solve specific problems, demonstrating the potential of AI to generate intelligent outputs. Since 1970s, researchers began developing early NLP systems, focusing on tasks like machine translation, speech recognition, and text generation. Systems like ELIZA (1966) and SHRDLU (1970) showcased early attempts at natural language generation.
2. Rule-based Systems and Neural Networks (1980s-1990s): Rule-based and expert systems continued to evolve during this period, with advancements in knowledge representation and inference engines. Neural networks, inspired by the structure of the human brain, gained attention in the 1980s. Researchers like Geoffrey Hinton and Yann LeCun made significant contributions to the development of neural networks, which are fundamental to GenAI.
3. Rise of Machine Learning (1990s-2000s): Machine learning techniques, including decision trees, support vector machines, and Bayesian networks, started becoming more prevalent. These methods laid the groundwork for GenAI by improving pattern recognition and prediction.
4. Deep Learning Resurgence (2010-2015): Deep learning, powered by advances in hardware and the availability of large datasets, experienced a resurgence in the 2010s. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) emerged as powerful tools for generative tasks such as image generation and text generation. Generative Adversarial Networks (GANs), introduced by Ian Goodfellow and his colleagues in 2014, revolutionized GenAI. GANs introduced a new paradigm for training generative models by using a two-network adversarial framework. Reinforcement learning also made strides in GenAI, especially in game-related domains.
5. Transformers and BERT (2015-now): Transformers, introduced in a seminal paper titled “Attention Is All You Need” by Vaswani et al. in 2017, became the backbone of many state-of-the-art NLP models. Models like BERT (Bidirectional Encoder Representations from Transformers) showed remarkable progress in language understanding and generation tasks. GenAI has found applications in various domains, including natural language generation, image synthesis, music compo-

sition, and more. Chatbots, language models like GPT-3, and creative AI tools have become prominent examples of GenAI in software implementation.

2.3. Fundamentals on LLMs

LLMs are AI systems trained to understand and process natural language. They are a type of ML model that uses very deep artificial neural networks and can process vast amounts of language data. An LLM can be trained on a very large dataset comprising millions of sentences and words. The training process involves predicting the next word in a sentence based on the previous word, allowing the model to “lear” the grammar and syntax of that language. With powerful computing capabilities and a large number of parameters, this model can learn the complex relationships in language and produce natural-sounding sentences. Current LLMs have made significant progress in natural language processing tasks, including machine translation, headline generation, question answering, and automatic text generation. They can generate high-quality natural text, closely aligned with the content and context provided to them.

For an incomplete sentence, for example: “The book is on the”, these models use training data to produce a probability distribution to determine the most likely next word, e.g., “table” or “bookshelf”. Initial efforts to construct large-scale language models used N-gram methods and simple smoothing techniques [24] [25]. More advanced methods used various neural network architectures, such as feedforward networks [26] and recurrent networks [27], for the language modelling task. This also led to the development of word embeddings and related techniques to map words to semantically meaningful representations [28]. The Transformer architecture [29], originally developed for machine translation, sparked new interest in language models, leading to the development of contextualized word embeddings [30] and Generative Pre-trained Transformers (GPTs) [31]. Recently, a common approach to improving model performance has been to increase parameter size and training data. This has resulted in surprising leaps in the machine’s ability to process natural language.

Parameter-Efficient Fine-Tuning

In October 2018, the BERT Large model [32] was trained with 350 million parameters, making it the largest Transformer model ever publicly disclosed up to that point. At the time, even the most advanced hardware struggled

to fine-tune this model. The “out of memory” issue when using the large BERT model then became a significant obstacle to overcome. Five years later, new models were introduced with a staggering 540 trillion parameters [33], a more than 1500-fold increase. However, the RAM capacity of each GPU has increased less than tenfold (max 80Gb) due to the high cost of high-bandwidth memory. Model sizes are increasing much faster than computational resources, making fine-tuning large models for smaller tasks impractical for most users. In-context learning, the ability of an AI model to generate responses or make predictions based on the specific context provided to it [31], represents the ongoing advancements in the field of natural language processing. However, the context limitation of Transformers reduces the training dataset size to just a few examples. This limitation, combined with inconsistent performance, presents a new challenge. Furthermore, expanding context size significantly increases computational costs.

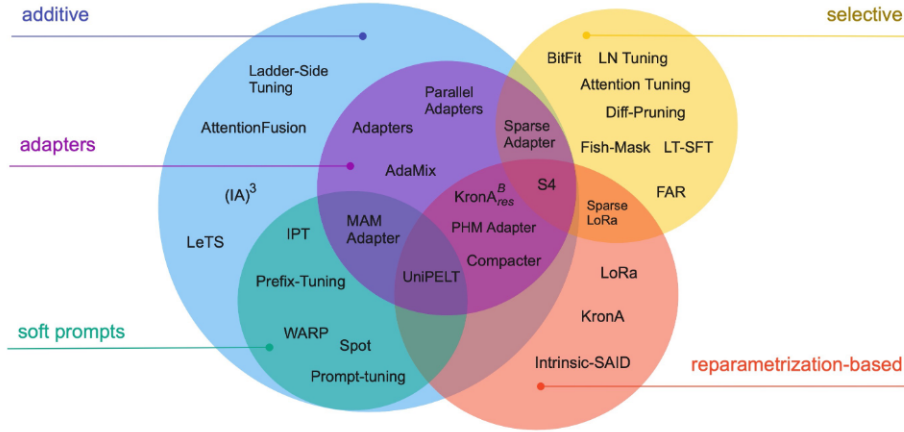


Figure 1: Several optimal parameter fine-tuning methods [34]

Parameter-Efficient Fine-Tuning methods have been researched to address these challenges by training only a small subset of parameters, which might be a part of the existing model parameters or a new set of parameters added to the original model. These methods vary in terms of parameter efficiency, memory efficiency, training speed, model output quality, and additional computational overhead (if any).

3. Research Approach

As indicated earlier, this paper aims to present a research agenda listing open Research Questions regarding GenAI in SE. We conducted a literature review (Section 3.1) and a series of focused groups to achieve this outcome (Section 3.2 and Section 3.3). The overall research process is presented in Figure 2.

3.1. Literature review

A comprehensive and systematic review might not be suitable for research on GenAI in software development at the time this research being conducted. Firstly, research work is rapidly conducted and published on the topic, hence rendering the findings of a comprehensive review probably outdated shortly after publication. Secondly, we found a lot of relevant work as gray literature from non-traditional sources such as preprints, technical reports, and online forums. These sources may not be as rigorously reviewed or validated as peer-reviewed academic papers, making it difficult to assess their quality and reliability. Thirdly, we would like to publish the agenda as soon as possible to provide a reference for future research. A systematic literature review would consume extensive effort and time, which might be obsolete by the time the review is complete.

Our strategy is to conduct focused, periodic reviews to capture the most current and relevant information without the extensive resource commitment of a comprehensive review. This approach allows for agility in keeping up with the latest developments without claiming comprehensiveness and repeatability. Our search approach involves two channels:

- Online portals: we used Google Scholar and Scopus to search with a formulated search string
- Gray literature sources: we search for papers from Arxiv and Paper-withCode
- Forward and backward snowballing: we scan citations forward and backward from the articles included in our review.

We used the search terms in Google Scholar and achieved the result (latest searched date October 2023). Google Scholar has the advantages of comprehensiveness and free access. It also covers grey literature where we found a significant number of research on GenAI at the searching time.

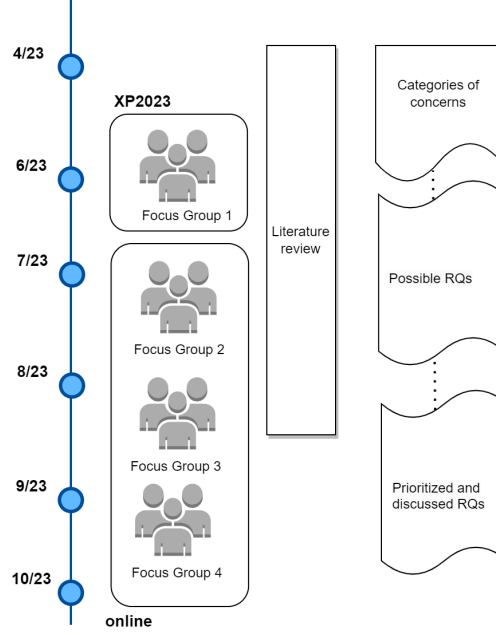


Figure 2: Research Agenda on GenAI for Software Engineering

3.2. Focus groups

We conducted four structured working sections as focus groups to identify, refine, and prioritize Research Questions on GenAI for SE. Focus groups have been used as a means to gather data in SE research [35, 36, 37, 38]. Focus group sessions produce mainly qualitative information about the objects of study. The benefits of focus groups are that they produce insightful information, and the method is fairly inexpensive and fast to perform. This differs from a brainstorming session where participants are selected and navigated by a moderator, following a structured protocol so that the discussions stay focused [36, 39]. Kontio et al. also suggest online focus groups with a lot of advantages, such as group thinking, no travel cost, anonymous contribution, and support for large groups [36]. The value of the method is sensitive to the experience and insight of participants. Hence, we presented the information of the participants in Table 1.

Focus groups' timeline is from April 2023 to September 2023 (as shown in Figure 2). All participants are SE researchers who have experience or interest in the topic, as shown in Table 1. For each focus group, we developed

Table 1: Demographic information of participants in workshops

Id	Background	Relevant Experience	No of focus groups
P01	Dr, Asst. Prof. in Software Engineering	Adopt ChatGPT in Agile context	4
P02	Prof. in Software Engineering and Applied AI	3+ years research and teaching on AI and SE	4
P03	Dr in Software Engineering	Research about GenAI in Agile context	4
P04	Dr in Software Engineering	5+ year research about NLP, applied AI in requirements engineering	3
P05	Dr in Software Engineering	Adopt ChatGPT in Agile context	4
P06	Prof. in Software Engineering and Applied AI	5+ years research and teaching on AI and SE	2
P07	Dr. in Software Engineering	Adopt ChatGPT in Agile context	2
P08	Asst. Prof. in Software Engineering	Pionner researchers in ChatGPT for Software Engineering	2
P09	Dr in Software Engineering	Research and conducted Systematic Literature Review on GenAI for SE	4
P10	Dr. in Software Engineering	Research on Applied AI	2
P11	Dr. in Software Engineering	Adopt ChatGPT in teaching Software Engineering classes	4
P12	Prof. in Software Engineering and Applied AI	5+ years research and teaching on AI and SE	4
P13	Prof. in Software Engineering and Applied AI	10+ years research and teaching on AI/ML for SE	2
P14	Dr. in Software Engineering, Certified Scrum Master	10+ years working in Agile projects, adopting ChatGPT in professional work	4
P15	Dr. in Software Engineering	Research and teaching Applied AI in SE	2

a plan, which included the agenda of the section and a set of exercises for the participants. Each focus group lasted between 2 to 3 hours. To capture data from focus groups, we use moderator’s notes, Miro boards ⁷, and recording in case of online sessions.

The focus of each focus group is different:

- Focus group 1 (Exploratory brainstorming): Brainstorming was aimed at exploring ideas for potential opportunities and challenges when adopting GenAI in software development activities. We discussed a question, “Which SE areas will benefit from GenAI tools, such as ChatGPT and Copilot?”. We used SWEBOK areas to initialize the conversations. At the end of the group, 11 categories were created. Each category is assigned a section leader who is a co-author responsible for synthesizing content for the section.
- Focus group 2 (Exploratory brainstorming): We discussed “What would be an interesting research topic for GenAI and Software Development and management?”. We initiated the discussion about “What could be interesting Research Questions for an empirical study on GenAI in SE?”. Some questions were generated during the working session.
- Focus group 3 (Validating brainstorming): prior to the meeting, a list of possible RQs was made. The list included 121 RQs. From this meeting, we aimed to validate the questions, if they are correctly and consistently interpreted, and if they are reasonable and meaningful to all participants of the meeting. Participants had sufficient time to review thoroughly and critically the RQs list. Each person left a comment for every question about their meaningfulness and feasibility. After this step, the RQs were revised and restructured.
- Focus group 4 (Validating brainstorming): The discussion was conducted in subgroups, each group focused on one particular SE category. Prior to this session, the question list was finalized. In total, there are 78 RQs in 11 categories. RQs that are consensusly not practically important or meaningful are excluded. We also discussed and ranked RQs according to their novelty and timeliness of each RQs. However,

⁷<https://miro.com>

Step 1: Move the notes below to the SE areas you think can gain benefits from GenAI tools

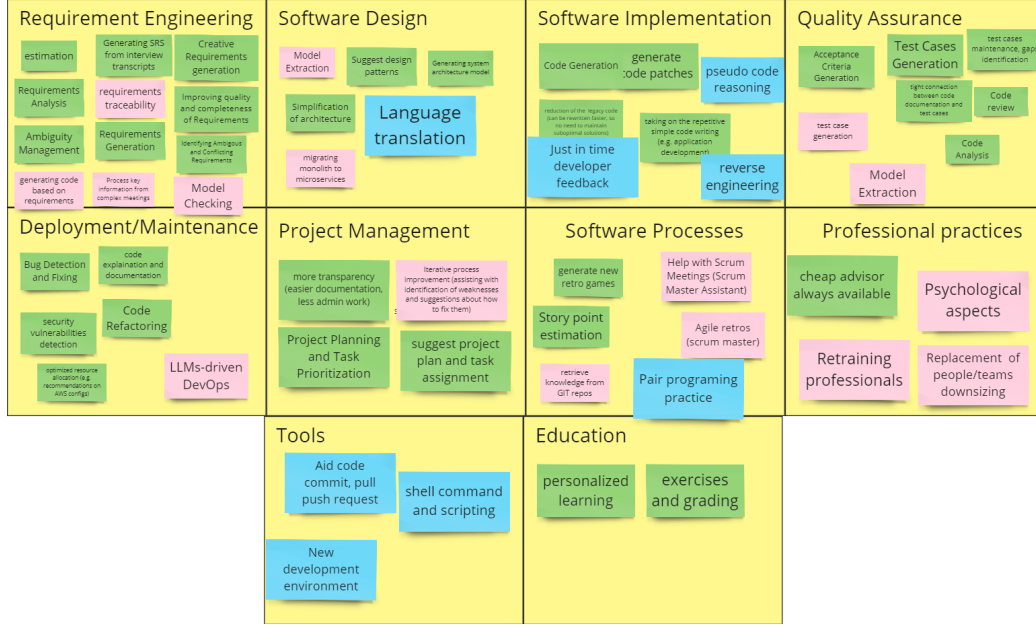


Figure 3: An outcome from Focus Group 2

we did not achieve consensus and a complete list of rankings for all RQs. Therefore, we decided not to present the ranking in this work.

3.3. Threats to validity

According to Kontio et al. [37, 36] threats of the focus groups method in SE studies include group dynamics, social acceptability, hidden agenda, secrecy, and limited comprehension. To minimize the group dynamics threat, i.e., uncontrollable flow of discussion, all sections were coordinated by the main author, strictly following the pre-determined agenda with time control. Social acceptability can influence discussion results, i.e. participants contribute biased opinions because of positive or negative perceptions from others. We introduce clear rules for contribution at the beginning of focus groups. For online sessions, Miro is used so participants can anonymously provide their opinions. As time is relatively limited for an online focus group, some complex issues or points might not be not necessarily understood by all participants (limited comprehension). This threat is mitigated by organizing

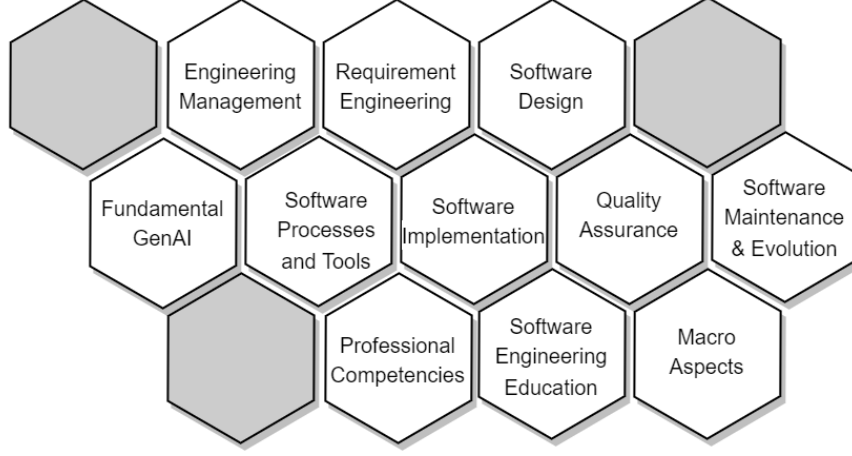


Figure 4: Research Agenda on GenAI for Software Engineering

focus groups following up with each other. Between the sessions, participants were required to do their "homework" so discussing points could be studied or reflected further beyond the scope of the focus groups. Four focus groups occurring in four months give sufficient time for individual and group reflection on the topic.

4. Research Agenda

We grouped the research concerns and questions into eleven tracks based on the thematic similarities and differences of the questions. While this grouping is one of the several possible ways to create the categories, it served the purpose of easing the presentation and discussion of the research agenda, shown in Figure 4. For each category, we present (1) its historical context, (2) key RQs, (3) State-of-the-art, (4) current challenges and limitations, and (6) future prospects.

4.1. GenAI in Requirements Engineering

4.1.1. Historical Context.

Requirements Engineering (RE) plays a pivotal role in the success of software projects. Although numerous techniques exist to support RE activities, specifying the right requirements has remained challenging as re-

quirements are seldom ready for collection in the initial stages of development [40]. Stakeholders rarely know what they really need, typically having only a vague understanding of their needs at the beginning of the project. These needs evolve during the project largely due to the dynamism inherent in business and software development environments [41]. Among Agile projects, RE has new challenges, such as minimal documentation, customer availability, lack of well-understood RE practices, neglecting non-functional requirements, lack of the big picture, inaccurate effort estimation and lack of automated support [42, 43]. Moreover, several other avenues in RE have remained relatively under-explored or under-developed, even in agile. These include, among others, the traceability and maintenance of requirements, stringent validation and verification processes, coverage of human factors in RE, ethical considerations while employing automation to RE or RE for AI software systems [44], coverage of domain specificity in requirements (e.g., stringent regulatory compliance for healthcare domain). RE, replete with its historical challenges and contemporary complexities, will need to adapt and evolve its role to the landscape of modern software development with technologies like GenAI, which alleviates some of these challenges [45].

4.1.2. Key RQs.

Regardless of the order in which RE tasks are performed or their application in either a waterfall or agile framework, the foundational RE stages and corresponding tasks remain consistent. This means that one still needs to identify stakeholders, elicit requirements, specify them in a format of choice, e.g., as user stories or shall-style requirements, analyze them, and perform other V&V tasks using them. Based on these considerations, we identified the following RQs.

1. How can GenAI support requirements elicitation?
2. How can GenAI effectively generate requirements specifications from high-level user inputs?
3. How can GenAI facilitate the automatic validation of requirements against domain-specific constraints and regulations?
4. How can GenAI be used to predict change requests?
5. What are the challenges and threats of adopting GenAI for RE tasks in different RE stages (pre-elicitation, elicitation, specification, analysis)?

4.1.3. *State-of-the-art*

Many research endeavors seek to utilize GenAI in order to automate tasks such as requirements elicitation, analysis, and classification. The current literature is dominant by preliminary empirical evidence of the significant role that LLMs in RE [46, 47, 48, 49, 50, 51, 52]. For instance, White et al. [46] introduced a catalog of prompt patterns, enabling stakeholders to assess the completeness and accuracy of software requirements interactively. Ronanki et al. [47] further substantiated this by comparing requirements generated by ChatGPT with those formulated by RE experts from both academia and industry. The findings indicate that LLM-generated requirements tend to be abstract yet consistent and understandable.

In addition to requirements elicitation, there's also a focus on requirements analysis and classification. Zhang et al. [53] developed a framework to empirically evaluate ChatGPT's ability in retrieving requirements information, specifically Non-Functional Requirements (NFR), features, and domain terms. Their evaluations highlight ChatGPT's capability in general requirements retrieval, despite certain limitations in specificity. In agile RE practice, Arulmohan et al. [49] conducted experiments on tools like Visual Narrator and GPT-3.5, along with a conditional random field (CRF)-based approach, to automate domain concept extraction from agile product backlogs. Their work provides evidence of the advancements in automating domain concept extraction. Both Ezzini et al. [2022] [50] and Moharil et al. [2023] [51] have tackled issues pertaining to anaphoric ambiguity in software requirements. Anaphoric ambiguity emerges when a pronoun can plausibly reference different entities, resulting in diverse interpretations among distinct readers [50]. Ezzini et al. [2022] [50] have developed an automated approach that tackles anaphora interpretation by leveraging SpanBERT. Conversely, Moharil et al. [2023] [51] have introduced a tool based on BERT for the detection and identification of ambiguities. This tool can be utilized to generate a report for each target noun, offering details regarding the various contexts in which this noun has been employed within the project. Additionally, De Vito et al. [54] proposed the ECHO framework, designed to refine use case quality interactively with ChatGPT, drawing from practitioner feedback. The findings indicate the efficacy of the approach for identifying missing requirements or ambiguities and streamlining manual verification efforts. BERT-based models have also been applied in RE to improve the understanding of requirements and regulatory compliance via automated question-answering [55, 56].

Hey et al. [2020] [57] introduced NoRBERT, which involves fine-tuning BERT for various tasks in the domain of requirements classification. Their findings indicate that NoRBERT outperforms state-of-the-art approaches in most tasks. Luo et al. [2022] [58], in a subsequent study, put forward a prompt-based learning approach for requirement classification using a BERT-based pre-trained language model (PRCBERT). PRCBERT utilizes flexible prompt templates to achieve accurate requirements classification. The conducted evaluation suggests that PRCBERT exhibits moderately superior classification performance compared to NoRBERT. Additionally, Chen et al. [59] evaluated GPT-4’s capabilities in goal modeling with different abstraction levels of information, revealing its retained knowledge and ability to generate comprehensive goal models.

4.1.4. *Challenges and Limitations*

Automation of RE tasks with GenAI techniques is undoubtedly promising; however, it is paramount for practitioners to be aware of several risks to ensure a judicious blend of domain and RE expertise and AI’s capabilities. Hallucination is a common challenge for adopting GenAI for all RE activities. GenAI techniques might generate requirements that appear sound but are either superfluous, incorrect, or inconsistent in a given domain context. Such inconsistencies can inadvertently lead to projects straying off course if not meticulously reviewed by stakeholders with relevant experience. Furthermore, some RE tasks (e.g., traceability) would require fine-tuning LLMs to the task data, but historically, not much RE data is publicly available to fine-tune these models accurately. We present some challenges and limitations of GenAI models regarding each RE stage, as below: **Pre-Elicitation Stage:** While LLMs can use preliminary research materials such as existing systems, universal documents, and app reviews to automate domain and stakeholder analysis, they sometimes fail to capture the broader context. Inaccurate identification of key domain concepts or overlooking intricate relationships can lead to suboptimal domain models.

Elicitation Stage: LLMs can process interview transcripts, survey responses, and observation notes to aid in the requirements elicitation process. However, the inherent biases of LLMs can affect the coding, analysis, and summarization of this data, potentially leading to skewed requirements.

Specification Stage: LLMs offer automated generation and classification of requirements. However, these generated requirements can sometimes lack clarity or coherence without proper human oversight. The reliance on

LLMs for regulatory compliance or requirements prioritization might also result in overlooking important nuances in legal or organizational guidelines.

Analysis Stage: Using LLMs for requirements quality analysis, traceability, and change management can automate and streamline the RE process. However, LLMs might sometimes struggle with intricate defect detection, traceability challenges, or handling complex change scenarios, leading to gaps in the analysis.

Validation Stage: Generative models often lose the broader context, especially with intricate requirements, potentially leading to discrepancies or conflicts within the generated requirements. The onus then lies on SE teams to validate such AI-generated requirements or other automated outputs, introducing an additional layer of complexity and potential validation challenges.

Overall, the ethical and security ramifications of delegating pivotal RE tasks to AI also warrant attention, mainly in safety-critical systems or application domains such as healthcare, defense, or cybersecurity. Lastly, a challenge that spans beyond just RE is the potential bias and fairness issues. If unchecked, generative models can perpetuate or even intensify biases in their foundational data. This can lead to skewed requirements that might not represent the diverse needs of end-users [60].

4.1.5. Future Prospects.

The future holds intriguing prospects regarding automating most RE tasks and making requirements engineers' lives easier. We believe that the future way of working will be AI-human collaborative platforms where GenAI assists domain experts, requirements engineers, and users in real-time, allowing for instantaneous feedback loops and iterative refining of requirements might be the new norm [61]. Such platforms could also offer visualization tools that help stakeholders understand how the AI interprets given inputs, fostering a more transparent understanding of system requirements and the generated outputs. We expect to witness LLMs capable of deeper contextual understanding, reducing inaccuracies in the pre-elicitation and elicitation stages. An important prospect would be allowing fine-tuning of models without exposing sensitive information from requirements (i.e., data leakage). Considering the ethical and safety concerns, there is an evident need for establishing robust frameworks and guidelines for the responsible use of AI in RE. In the near future, the research could be directed at creating models with built-in ethical considerations capable of addressing the known biases,

ensuring that the generated requirements uphold the standards and are truly representative and inclusive.

4.2. GenAI in Software Design

4.2.1. Historical Context

Design and architecture decisions are at the core of every software system. While some well-known structures are easy to replicate and create templates, some decisions need an evaluation of the trade-offs involved, especially on how these decisions affect the system's quality attributes. In this context, choosing the pattern or solution is just the last step of the process, which requires identifying all factors that influence and might be affected by the decision. We can find in the literature several works that proposed solutions to automate design decisions, mainly based on the choice of patterns, such as based on questions and answers [62], based on anti-patterns detection [63], based on text classification [64], and based on ontologies [65]. However, it is a fact that these approaches are not yet widely used in industry, and GenAI is attracting interest from professionals as a promising approach to be adopted for design decisions in real projects.

4.2.2. Key RQs

Similar to Requirements Engineering, it is necessary to explore the possible adoption of GenAI in all software design activities. As listed below, we present RQs addressing specific software design activities to illustrate open challenges in GenAI and software design:

1. How can GenAI assist in identifying and selecting appropriate design and architectural patterns based on requirements and constraints?
2. What strategies can be adopted to promote collaboration for decision-making between professionals and GenAI in the software design process?
3. What are the limitations and risks associated with using GenAI in software design, and how can these be mitigated to ensure their reliability and validity?
4. How can GenAI be employed in a continuous software design process, automating the identification of improvements and refactoring to optimize key quality attributes?
5. How can generative AI be utilized to automate the design and implementation of user interfaces, improving user experience in software applications?

6. How can GenAI optimize the trade-offs between different quality attributes, such as performance, scalability, security, and maintainability?
7. What strategies can be employed to enable GenAI to adapt and evolve system architectures over time, considering changing requirements, technologies, and business contexts?

4.2.3. State-of-the-art

We do not find many papers exploring GenAI or LLM for software design activities. Ahmad et al. described a case study of collaboration between a novice software architect and ChatGPT to architect a service-based software [66]. Herold et al. proposed a conceptual framework for applying machine learning to mitigate architecture degradation [67]. A recent exploratory study evaluated how well ChatGPT can answer that describes a context and answer which design pattern should be applied [68], demonstrating the potential of such a tool as a valuable resource to help developers. However, in the cases where ChatGPT give a wrong answer, a qualitative evaluation pointed out that the answer could mislead the developers. Stojanović et al. presented a small experiment where ChatGPT identified microservices and their dependencies from three pieces of system descriptions [69]. Feldt et al. proposed a hierarchy of design architecture for GenAI-based software testing agents [70].

4.2.4. Challenges and Limitations

The challenges of GenAI in software design go beyond just receiving the correct answer for a design problem. It is essential to understand what role it should play and how it will interact with the software developers and architects in a team. Considering that design decisions happen at several moments during a project, it is also important to understand how a GenAI can be integrated into that process.

A limitation of the current literature on the topic is the lack of studies in real and industrial environments. While the techniques were experimented with in a controlled environment, it is still unknown how useful they can really be in practice. While studies show that the usage of AI can provide correct answers to software design problems with good accuracy, inside a real project environment, it is still a challenge to understand which information needs to be provided and if the answer can really be insightful and not just bring information that the team already knows. While the recent works

delimit the scope in a limited set of patterns, usually the classic Gang of Four (Gof) patterns [71], for industrial usage, answers are expected to consider a wide spectrum of solutions. Considering the findings of a recent study that exposed several problems in how non-functional requirements are handled [72], real projects might not have all the information needed as input for a GenAI tool to give an accurate answer.

4.2.5. Future Prospects

With access to tools that can give useful answers, the key challenge for the future of GenAI in software design is how GenAI tools will fit into the existing processes to enable its usage more consistently. Considering that the design of the architecture happens continuously, using these tools only as an oracle to talk about the solutions as a personal initiative might be an underutilization of its potential. To address the RQs in this domain, it is imperative to comprehend the roles GenAI tools can assume in the software design process and the approaches for engaging with them using prompt patterns, as discussed in [73]. It is expected that new good and bad software design practices will soon emerge from the first experiences with this kind of tool.

4.3. GenAI in Software Implementation

4.3.1. Historical Context

Software implementation is the crucial phase where the designed solution is transformed into a functional working application. With the advent of object-oriented programming in the 1990s, software systems have rapidly grown in size and complexity. Automation is an important research direction to increase productivity while ensuring the quality of software code. Modern Integrated Development Environments (IDEs) with features, i.e. compiling, deploying, debugging and even code generating, are continuously evolving to maximize the support for software developers.

However, implementing software systems in industries today presents several challenges. First, there is the constant pressure to keep up with rapidly evolving technology, which often leads to compatibility issues with legacy systems. Second, ensuring security and data privacy has become increasingly complex as cyber threats continue to evolve. Third, scalability and performance optimization are critical as software systems must handle growing amounts of data and users. Lastly, the shortage of skilled software engineers

and the rising cost of software development further exacerbate these challenges. Thus, successful software implementation requires careful planning, ongoing maintenance, and adaptability to stay ahead in today’s competitive technological landscape.

Code generation stands as a crucial challenge in both the field of GenAI and the software development industry. Its objective is to automatically generate code segments using requirements and descriptions expressed in natural language. Addressing this challenge can yield a multitude of advantages, including streamlining the programming process, improving the efficiency and accuracy of software development, reducing the time and effort required for coding, and ensuring the uniformity and correctness of the generated code. To showcase the capability of code generation, we requested ChatGPT with a simple task:” Create a function that takes two arrays as input, then performs the quick sort algorithm on each array and returns the results of the two sorted arrays with the phrase ‘Here you go’.” (Figure 5) To complex requests, such as a new algorithm problem taken from Leetcode. (Figure 6). The result is a complete solution that, when submitted, is correct for all test cases on Leetcode⁸.

4.3.2. Key RQs

Considering the challenges identified for modeling and training GenAI architecture in the field of Software Implementation, We highlight particular RQs (RQs) related to the process of building a model that can be used in a real-world industrial environment

1. To what extent GenAI-assisted programming, i.e. code summarization, code generation, code completion, comment generation, code search, and API search can be effectively integrated into practical software development projects?
2. How can GenAI models be specialized for specific software domains, such as web development, game development, or embedded systems, to generate domain-specific code more effectively?
3. How to ensure the correctness and reliability of generated results, carefully consider the potential for hidden malicious output that we are unaware of?

⁸<https://leetcode.com/>

4. How can software companies both leverage the capacity of LLMs and secure their private data in software development?
5. Does the outcome generated by a GenAI model have a significant bias towards the knowledge learned from LLMs while overlooking important input segments from private datasets of an enterprise?
6. How can the system effectively specify user intent when users communicate by progressively providing specifications in natural language
7. How can GenAI model be built, trained and retrained for a cost-based performance in various types of software implementation tasks? [74] [75]
8. What needs to be done to achieve a Natural Language Interface for Coding where non-IT people can also interact and develop their wanted software?
9. What methods can be employed to validate AI-generated code against functional and non-functional requirements?
10. How can we ensure GenAI models comply with certain legal and regulation constraints [76] [77] ?

4.3.3. State-of-the-art

Developing code in an efficient, effective, and productive manner has always been an important topic for SE research. Since 2020, there has been extensive research about LLMs for various coding tasks, such as code generation, code completion, code summarization, code search, and comment generation. Various studies investigate the code generation performance of GitHub Copilot in different work contexts and across different Copilot versions, e.g., [78, 79, 80]. Jiang et al. proposed a two-step pipeline that use input prompts to generate intermediate code, and then to debug this code [81]. The approach is promising to bring consistent efficacy improvement. Dong et al. treated LLMs as agents, letting multiple LLMs play distinct roles in addressing code generation tasks collaboratively and interactively [82]. Borji et al. presented a rigorous, categorized and systematic analysis of LLM code generation failures for ChatGPT [83]. Eleven categories of failures, including reasoning, factual errors, mathematics, coding, and bias, are presented and discussed in their work. Sun et al. focus on users' explainability needs for GenAI in three software engineering use cases: code generation based on natural language description (with Copilot), translation between different programming languages (with Transcoder), and code autocompletion (with Copilot) [84].

Several studies attempt to compare different code generation tools. Li et al. proposed an approach called AceCoder that surpasses existing LLM-based code generation tools, i.e. CodeX, CodeGeeX, CodeGen, and InCoder on several benchmarks [85]. Doderlein et al. experimented with various input parameters for Copilot and Codex, finding that varying the input parameters can significantly improve the performance of LLMs in solving programming problems [3]. Yetistiren et al. [74] presented a comprehensive evaluation of the performance of Copilot, CodeWhisperer, and ChatGPT, covering different aspects, including code validity, code correctness, code security, and code reliability. Their results show a wide degree of divergence in performance, motivating the need for further research and investigation [86]

Salza et al. presented a novel LLM-driven code search approach by pre-training a BERT-based model on combinations of natural language and source code data and fine-tuning with data from StackOverflow [87]. Chen et al. explored the setting of pre-train language models on code summarization and code search on different program languages [88]. Guo et al. developed a model named CodeBERT with Transformer-based neural architecture for code search and code generation [89]. The model has been widely popular and serves as a baseline for a lot of LLM-driven code generation approaches. For instance, Guo et al. proposed a model called GraphCodeBERT, a pre-trained model for programming language that considers the inherent structure of code. GraphCodeBert uses data flow in the pre-training stage and is textured on code search and code generation [90]. Code retrieval, a practice to reuse existing code snippets in open-source repositories [91], is also experimented GenAI. Li et al. proposed a Generation-Augmented Code Retrieval framework for code retrieval task [92].

Automatic API recommendation is also a sub-area of LLM applications. Wei et al. proposed CLEAR, an API recommendation approach that leverages BERT sentence embedding and contrastive learning [93]. Zhang et al. [387] developed ToolCoder, which combines API search tools with existing models to aid in code generation and API selection [94]. Patil et al. [95] developed a tool called Gorilla. This is a fine-tuned LLaMA-based model that is able to interact and perform with API, i.e. API call and verification.

Comment generation is also an active research area, as source code is not always documented, and code and comments do not always co-evolve- [96]. Mastropaolo et al. studied comment completion by validating the usage of Text-To-Text Transfer Transformer (T5) architecture in autocompleting a code comment [96]. Geng et al. adopted LLMs to generate comments

that can fulfill developers’ diverse intents [97]. The authors showed that LLM could significantly outperform other supervised learning approaches in generating comments with multiple intents.

Several works have been done in an industrial setting. Vaithilingam et al. [98] explored how LLMs (GitHub Copilot) are utilized in practice in programming, finding that even though they did not necessarily increase work performance, programmers still preferred using them. Barke et al. [99], in a similar study of how programmers utilize LLMs, highlighted two use modes: acceleration (next action is known and LLM is used to code faster) and exploration (next action is unknown and LLM is used to explore options). Denny et al. showed that Copilot could help solve 60 percent of programming problems and a potential way to learn writing code [2]. Ziegler et al. [100] explored potential measures of productivity of LLM use in programming, finding that the ”rate with which shown suggestions are accepted” is a metric programmers themselves use to gauge productivity when utilizing LLMs.

Ouyang et al. empirically studied the nondeterminism of ChatGPT in code generation and found that over 60% of the coding tasks have zero equal test output across different requests [5]. Pearce et al. discussed potential security issues related to code generated with GitHub Copilot [101]. Finally, in terms of programming practices, the use of LLM-based assistants such as GitHub copilot is often likened to pair programming [102, 79].

4.3.4. Relevant technologies

The flow of constructing the code generation model is depicted in Figure 7. As seen, the flow of the automatic code generation system is defined as follows:

- Step 1:** Research and utilize LLMs that have the best performance and an appropriate number of parameters for tasks related to automatic code generation and code summarization. In this case, the author employs two models: *CodeGen-6B-Mono* for code generation (Generation) and *CodeT5-Large* for code summarization.
- Step 2:** Implement methods to optimize parameter fine-tuning on *specific private datasets* or carefully select public datasets that were not part of the training set of the initial large models (downstream datasets). Here, the author uses the *LoRA method* and the HuggingFace Peft library⁵ to handle the two defined downstream tasks.

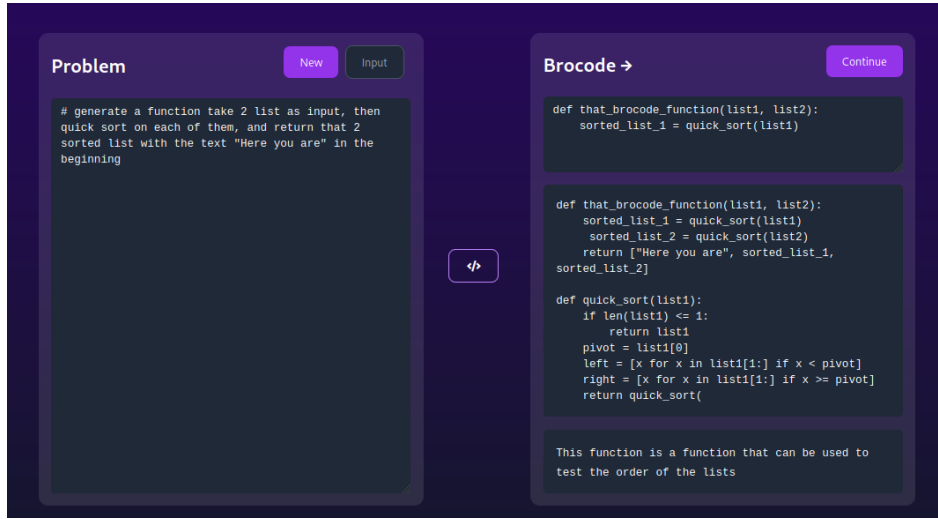


Figure 5: Results of the proposed model for simple logic requests

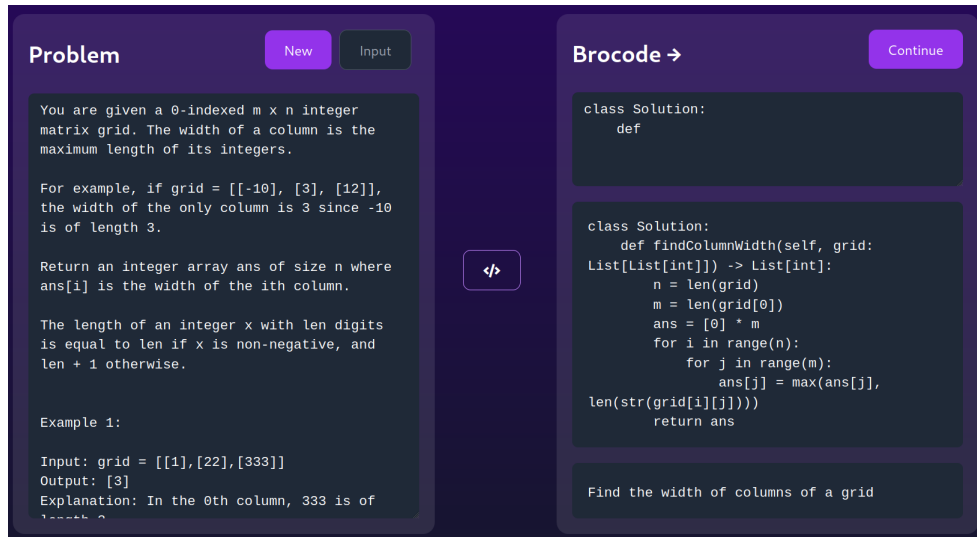


Figure 6: Results of the proposed model for complex requests

Step 3: Store the trained model and develop a *user interface*, enabling users to interact and experience automatic code generation and summarization based on the initial specific dataset.

Step 4: Develop secure storage mechanisms for user-specific projects and con-

Latest LLMs for Code Generation	Latest Parameter Fine-Tuning Methods
<ul style="list-style-type: none"> • CodeGen-Multi: <i>SalesForce</i> • CodeT5+: <i>SalesForce</i> • SantaCoder: <i>HuggingFace</i> • StarCoder: <i>HuggingFace</i> • Code Llama: <i>Facebook</i> • PaLM-Coder: <i>Google</i> 	<ul style="list-style-type: none"> • LoRA [103] • AdaLoRA [48] • Prefix Tuning [104] [105] • P-Tuning [106] • Prompt Tuning [107] • MultiTask Prompt Tuning [108] • $(IA)^3$ [109]

Table 2: Latest development in LLM training and fine-tuning

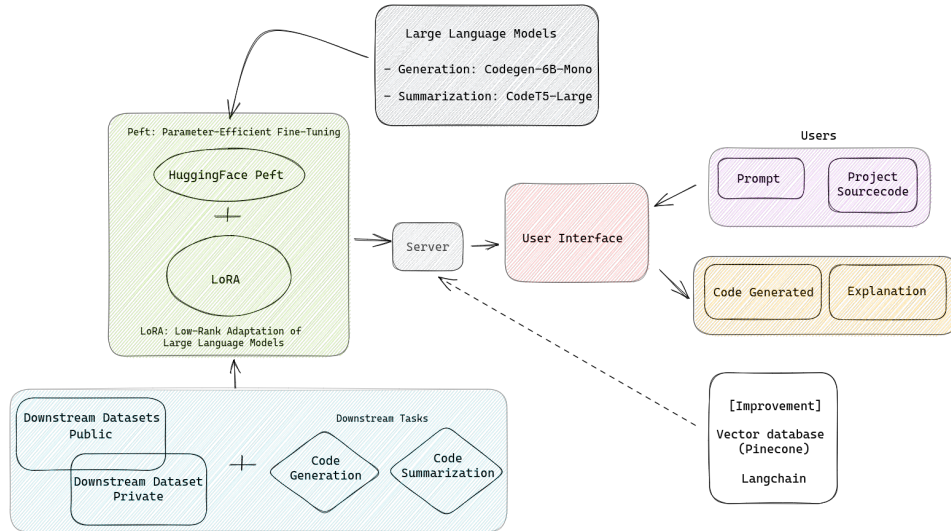


Figure 7: Overview of the GenAI for code

tinue training on new user datasets.

4.3.5. *Challenges and Limitations*

Challenges and Limitations of current GenAI applications on software implementation can be presented under three aspects: (1) expressiveness, (2) program complexity, and (3) security and specification requirements.

Heterogeneous inputs For an AI generation system to function effectively, the first unavoidable challenge is understanding the input requirements. The system needs to analyze and grasp specific requirements, determining what it needs to generate to satisfy those particular demands. Users' intentions can be expressed in heterogeneous manners, i.e. specifying logical rules, using input-output examples, or describing in natural language. Addressing all logical rules comprehensively is demanding, while input-output examples often don't sufficiently describe the entire problem. It can be argued that the ability to recognize and articulate user intentions is the key to the success of automated code generation systems, a challenge that traditional approaches have yet to solve.

Model Size With the increasing size and complexity of codebases and the emergence of numerous new technologies, understanding and generating code is also becoming more and more complex. GenAI model might need to include more and more dimensions, making it impractical and unrealistic for most people to fine-tune large models for smaller tasks. It's turning into a race dominated by big global tech corporations. Notable applications such as ChatGPT, Microsoft's Github Copilot, Google's Bard, and Amazon's Code Whisperer share a common goal: harnessing AI to make programming more accessible and efficient. This race is intensifying. In May 2023, HuggingFace launched its LLM BigCode [110], while Google announced its enhanced PaLM-2 model. Both made astonishing advancements, competing directly with Microsoft's famous GPT models. Recently, in August 2023, Facebook announced Code LLaMa, which is expected to give the open-source community easier access to this exciting race.

Security and Specification Requirements Security is also a major challenge for GenAI systems. Due to their vast and comprehensive knowledge base, they are in a position where they "know everything, but not specifically what." Alongside this, there's an increasing need for code security and very specific logical inputs in real-world scenarios. Consider not retraining the GenAI system with the code and knowledge of a specific (private) project but instead applying general knowledge to everyday tasks in the Software Industry. It can be easily predicted that the system won't be able to gener-

ate code satisfying complex real-world requirements. Applying the specific logic of each company’s security project requires the system to grasp not just general knowledge but also the business and specific logic of each particular project. However, suppose an organization or individual tries to retrain a LLM on their project’s source code. This is not very feasible. First, training an LLM requires robust hardware infrastructure that most individuals or organizations cannot guarantee. Secondly, the training time is usually significant. Lastly, the result of training is a massive weight set, consuming vast storage space and operational resources. Resources and efforts spent on training multiple different projects can be wastefully redundant, especially when users only need the model to work well on a few specific, smaller projects. On the contrary, focusing on training specific projects from the outset means we can’t leverage the general knowledge LLMs offer, leading to lower-quality generated code. Thus, the author believes that while automated code generation might work well for general requirements using *LLM*, there remain many challenges when applying them to specific and unique projects. To address these challenges, recent years have seen numerous studies dedicated to discovering methods to *fine-tune LLM* using parameter optimization methods. Most of these approaches share a goal: applying general knowledge to specific problems under hardware constraints while retaining comparable efficiency and accuracy.

4.3.6. Future Prospects

The intrinsic appeal of GenAI on a global scale is undeniable. Ongoing research endeavors are significantly focused on enhancing optimization techniques and LLMs to yield forthcoming generations of GenAI models characterized by heightened potency, user-friendliness, rapid trainability, and adaptability to specific projects or downstream applications. Adaptive GenAI-based tools, such as PrivateGPT ⁹, an enclosed ChatGPT that encapsulates comprehensive organizational knowledge, have surfaced and garnered substantial community support. Another tool, MetaGPT ¹⁰ illustrates a vision where a software development project can be fully automated via the adoption of role-based GPT agents.

For the first time in history, programming is becoming accessible to a

⁹<https://www.privategpt.io/>

¹⁰<https://github.com/geekan/MetaGPT>

broad spectrum of individuals, including those lacking a formal background in Software Engineering (SE). GenAI will offer opportunities to democratize coding, enabling individuals from diverse backgrounds to partake in software development. With its intuitive interface and robust automation capabilities, GenAI serves as a conduit for individuals to unlock their creative potential and manifest innovative concepts through coding, irrespective of their prior technical acumen. This inclusive approach to programming not only fosters a more diverse and equitable technology community but also initiates novel avenues for innovation and complex problem-solving within the digital era.

The current AI landscape parallels the challenges inherent in edge machine learning, wherein persistent constraints related to memory, computation, and energy consumption are encountered. These hardware limitations are being effectively addressed through established yet efficacious technologies such as quantization and pruning. Concomitant with these hardware optimizations, there is a notable rise in community involvement, affording opportunities for research groups beyond the purview of major technology corporations. This augurs the imminent development of groundbreaking GenAI products finely tailored for the software industry.

4.4. GenAI in Quality Assurance

4.4.1. Historical Context

Software Quality Assurance (SQA) activities are imperative to ensure the quality of software and its associated artifacts throughout the software development life-cycle. It encompasses the systematic execution of all the necessary actions to build the confidence that the software conforms to the implied requirements, standards, processes, or procedures and is fit for use[111]. The implied quality requirements include, but are not limited to, functionality, maintainability, reliability, efficiency, usability, and portability. Often, SQA and software testing are used interchangeably when software quality is concerned. However, both these terms are intricately intertwined[111]. While SQA is a broader domain, testing remains a key and integral part of it, executing the software to detect defects and issues associated with quality requirements. Testing is widely practiced in the software industry despite the fact that it is considered an expensive process.

The testing process begins with understanding the requirements and generating test case requirements thereafter. Depending on the software system under test, software development life-cycle in practice and implied requirements, various types of tests are usually conducted. These include but are

not confined to (functional) unit testing, integration testing, system testing, acceptance testing, and finally, testing the (required) quality attributes as per the non-functional requirements.

Through a recent and extensive survey[112], several key challenges related to SQA and testing are disseminated by the SQA community. *Development of automated tests, limited time, budget, and human resources, inadequacies of SQA training, adapting SQA to contemporary and evolving software development workflows, and understanding requirements before coding* are among the key challenges. A similar list of challenges is presented in another study[113]. It reports that traditional methods of testing, and test case design consume a significant amount of energy, time and human resources. For Yuan et al.[114], generating tests manually, according to the existing practices, is laborious and time-consuming. As a result, the efficiency of conducting these activities becomes low while the extent and intensity of these activities also rely on the experience of the SQA professionals[113].

These challenges make it hard to meet the increasing demand for SQA and testing across software companies. On the other hand, recent advancements in GenAI, especially in automatic code generation through LLMs, demand a significant transformation in how SQA functions within the current paradigm. Reduced time and resources required to perform SQA functions by automating various activities are among the evident benefits of utilizing GenAI for SQA. In fact, one of the existing obstacles faced by the SQA community is the flaws in the test data that make automated test sets fail for software systems[112]. Utilizing GenAI in SQA to generate diverse test data based on specified requirements, scenarios, and use cases effectively addresses these challenges. Moreover, GenAI has the potential to generate insights from the analysis of existing test data, enabling the SQA community to produce more reliable software systems effectively and efficiently.

4.4.2. Key RQs

In light of the challenges faced by the SQA community, we propose several key questions for using GenAI in software quality. The questions include:

1. To what extent GenAI can replace developers in code reviews?
2. Can GenAI help to produce test cases and test scenarios from Software Requirements Specification (SRS)?
3. Are there common error patterns of AI-generated code?
4. How GenAI can be integrated effectively into quality assurance pipelines?

5. How can GenAI be utilized to automate acceptance criteria from high-level requirements?
6. How can GenAI be used to aid the creation of diverse and effective test suites for software testing?
7. How can GenAI assist in maintaining and updating test cases in case of requirement changes?
8. What strategies can be employed to ensure that the generated test cases by GenAI are reliable, effective, and representative of real-world usage scenarios?

As a software quality assurance engineer, your role is to create a set of tests to make sure our login system works correctly. This system includes fields for both the username and password. Here are the rules we need to follow:

 Usernames should be made up of letters and numbers, and they must be between 5 and 10 characters long.

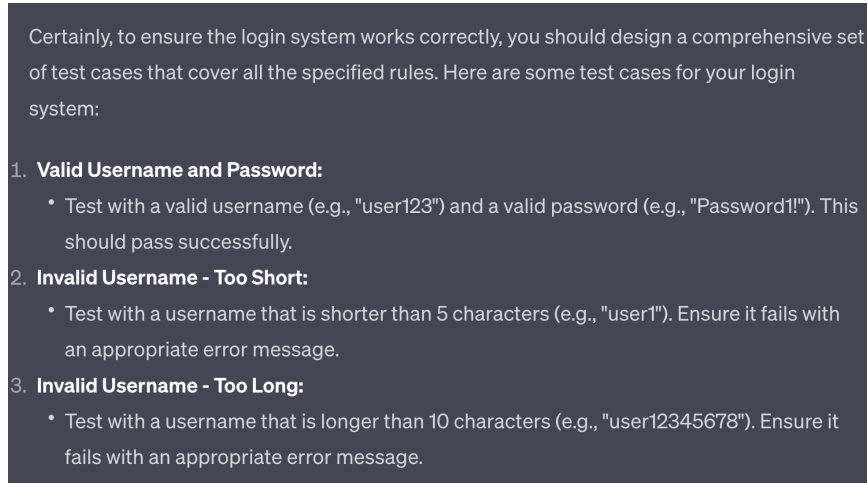
 Passwords need to be at least 8 characters in length. They must contain both a letter and a number or a special character. Also, the username cannot be the same as the password.

 Your job is to design tests that cover these requirements and ensure our authentication system functions as intended.

Figure 8: A Sample Prompt to Generate Test Cases Based on Requirements

4.4.3. *State-of-the-art*

Considering the challenges faced within this domain, a large number of studies have been conducted to explore, propose and evaluate different QA approaches. The adoption of AI also has a long history in software quality research. Primarily, AI has been utilized in automating and optimizing tests, predicting faults, and maintaining software quality[115]. With the recent advancement in AI models, GenAI might support various SQA activities. However, existing GenAI models (e.g. ChatGPT) are not particularly trained to perform these specialized jobs. Thus, they might offer limited performance when applied to particular situations[116]. Nevertheless, these recent breakthroughs in the field have invited several researchers to find possible ways to address the challenges faced by the SQA community and tailor and evaluate existing models for SQA tasks. Regarding this, we report a



Certainly, to ensure the login system works correctly, you should design a comprehensive set of test cases that cover all the specified rules. Here are some test cases for your login system:

- 1. Valid Username and Password:**
 - Test with a valid username (e.g., "user123") and a valid password (e.g., "Password1!"). This should pass successfully.
- 2. Invalid Username - Too Short:**
 - Test with a username that is shorter than 5 characters (e.g., "user1"). Ensure it fails with an appropriate error message.
- 3. Invalid Username - Too Long:**
 - Test with a username that is longer than 10 characters (e.g., "user12345678"). Ensure it fails with an appropriate error message.

Figure 9: Part of the Generated Test Cases (Response)

number of studies that are aimed at understanding and evaluating the use and effectiveness of GenAI models in specialized settings.

Jalil et al. [116] explored the use of GenAI to educate novice software test engineers. The authors reported that LLMs, like, for instance, ChatGPT have been able to respond to 77.5% of the testing questions while 55.6% of the provided answers were correct or partially correct. The explanations to these answers were correct by the of 53%. With these percentages in mind, it has been ascertained that prompting techniques and additional information in queries might enhance the likelihood of accurate answers[116]. Ma et al.[117] agreed with this by pointing out that the prompt design significantly impacts the performance of GenAI models. In this context, White et al. [118] proposed various prompt patterns to improve code quality and perform testing while interacting with GenAI models.

Akbar et al. [119] reported the possible uses of GenAI models for software quality and testing. Yuan et al.[114] showed the potential of LLMs in unit test generation, which is among the most popular test phases in testing after model-based testing [120]. Similarly, it is also ascertained that GenAI models can be optimized to simulate various testing activities like, for instance, testing usability-related requirements of software systems[119]. As a result, software companies can save time and resources[116][114] and simultaneously improve the software quality and performance[119]. In addition, based on the literature survey, the study[119] also identifies several factors

that can make an organization adopt GenAI models for testing and software quality. These factors include the generation of software test data, the capability to fine-tune these models for SQA tasks, generating test scenarios, bug reporting, automated reporting on the performance and software quality, and the tailored use of these models as testing assistants. Keeping these enablers in mind, we observed that the current literature provides no clear guidelines on how to achieve the objectives of using GenAI in SQA.

Liu et al.[113], in their recent study, attempted to leverage the capabilities of ChatGPT for testing purposes and developed a system for automated test case generation using ChatGPT. They report interesting comparisons between manually constructing test suites and generating them through ChatGPT. According to the experimental results of the study [113], the efficiency in generating test suits through their ChatGPT-based system exceeds 70%. The new system uses ChatGPT as an intelligent engine to do the job. In contrast to Liu et al.[113], Yuan et al. [114] claimed that only 24.8% of tests generated by ChatGPT were executed successfully. The remaining tests suffered from correctness issues[114][121]. Most of these issues include compilation errors (57.9%) followed by incorrect assertions (17.3%). While on the other hand, Yuan et al. [114] found that the tests generated by ChatGPT are similar to human-generated tests with a sufficient level of readability and cover the highest test coverage.

While discussing the scope of ChatGPT for code analysis, Ma et al. [117] used a significant number of codes and concluded that ChatGPT is great at understanding code syntax. However, it fails to fully comprehend dynamic semantics. The experimental evaluation of this study also demonstrates the GenAI model’s limited ability to approximate dynamic behaviors.

Lastly, in a large-scale literature review, Wang et al. [121] collected and analyzed 52 empirical and experimental studies on the use of GenAI LLMs for software testing. The authors reported commonly employed LLMs, and software testing tasks that can be accomplished through LLMs. In this regard, test case preparation and program repair remained the two most commonly explored areas of prior investigations[121]. Other areas include test oracle generation, bug analysis, debugging, and program repair. On the contrary, CodeX, ChatGPT, codeT5, CodeGen and GPT-3 are commonly utilized models by researchers to verify and validate particular but a few aspects of the software quality.

4.4.4. *Challenges and Limitations*

Like software implementation, the utility of GenAI models for SQA activities is fully potential, but also challenging. The above studies show the illustrative use of GenAI in assessing code quality, analyzing both human and AI-generated code, conducting code reviews, and generating tests and test data. Nevertheless, inadequacies of current models do exist e.g. to assess the dynamic behavior of the code, human intervention to identify the hallucination, when dealing with code semantic structures and non-existent codes. Therefore, despite the promising benefits that GenAI offers to SQA, there is a need to evolve the existing methodologies, tools, and practices. We present some of the challenges and limitations of GenAI for SQA extracted from the literature review:

- The tests generated by GenAI models currently demonstrate issues related to correctness, hallucination, and a limited understanding of code semantics[113][117][122]. These challenges are particularly notable in comprehending the dynamic behavior of the code. In addition, these challenges highlight the need for tremendous training data to refine models to perform specialized SQA tasks[119][?]. The training requires investing a significant amount of resources, which might not be a feasible option for all software development companies[123]. These issues make the limited practicality of GenAI models in their current state, within the SQA context.
- Biasness in the training data may also affect the potential behavior of GenAI models towards SQA activities[119]. As a result, human intervention for manual correction or identification of vulnerabilities remains mandatory. Alongside this, biases in the training data might raise ethical issues or lead to a negative impact on software quality.
- Output of GenAI models is highly associated with the prompting techniques[?][118]. Providing more information or additional context of testing or quality might result in a better or improved answer. It further requires researchers to design proper prompt engineering techniques for SQA professionals.
- Almost all of the existing studies on the topic(e.g. [?][113][117][114]) are mostly experimental studies and thus do not take into consideration the industrial context. Therefore, how GenAI models deal with

real-world software quality issues remains a mystery. We need more real-world case studies and industrial examples to understand the effectiveness of various tasks of SQA with GenAI. Moreover, it would be interesting to study how well GenAI enhances the productivity of SQA professionals.

4.4.5. *Future Prospects*

The existing studies stimulate the potential of utilizing GenAI in assessing software quality and thus demand reshaping the traditional SQA implementation methods, practices, and tools. Shortly, we foresee a significant number of studies within this domain. Therefore, based on the existing understanding of the use of GenAI in this domain and considering the challenges and limitations faced, we propose several possible avenues for forthcoming research studies:

Exploring GenAI for Further SQA Tasks: Future research might investigate conducting and evaluating early activities of SQA through GenAI. This includes creating a complete test plan or generating test requirements based on SRS. However, automating this process would still require a human expert to validate the outcome because these activities rely heavily on domain knowledge. The alternative way to achieve this could be training the GenAI models with relevant domain data[121].

Likewise, much of the existing literature studied functional testing followed by security testing in terms of finding vulnerabilities and asking for bug repair[121]. We found only one study [119] that alludes to the use of GenAI to test usability requirements. However, even in this case, the practices are not obviously described. Similarly, other areas of testing are not currently the focus of existing studies, i.e. acceptance testing, integration testing, and testing other software quality requirements like performance, reliability, efficiency, and portability. Therefore, future research on these SQA activities would be useful to show the effectiveness and limitations of existing GenAI models.

Exploring the Impact of Prompt Strategies :Further research is also required to evaluate the effect of prompt strategies on GenAI to accomplish particular SQA tasks. The existing studies only employed four prompt engineering techniques so far, out of 12 [121]. The effect of prompting on the GenAI models is also acknowledged by other studies e.g. [118][117][119]. Therefore, a future prospect might be to identify effective prompting strategies for a particular task, such as unit test code generation.

Utilizing Latest GenAI Models: Wang et al. [121] found that Codex and ChatGPT are two largely explored GenAI models to assess the usefulness of GenAI for SQA activities. However, as expected, we do not find any study utilizing the latest version of ChatGPT i.e. ChatGPT 4.0. Therefore, investigating the use of the latest GenAI model might produce different performance results for varying SQA activities. Using a different dataset and a different GenAI model would offer a different effect on the performance results.

4.5. GenAI in Maintenance and Evolution

4.5.1. Historical Context

Software maintenance encompasses all the activities involved in managing and enhancing a software system after it has been deployed, ensuring the system's continued quality in responding to changing requirements and environments [124]. As software maintenance is an expensive activity that consumes a considerable portion of the cost of the total project [125, 124], software systems should be developed with maintainability in mind. Unfortunately, software maintainability is frequently neglected, resulting in products lacking adequate testing, documentation, and often being difficult to comprehend. Therefore, to maintain a software system, one must usually rely on the knowledge embodied and embedded in the source code [124]. In practice, software engineers spend a lot of time understanding code to be maintained, converting or translating code from one environment to another, and refactoring and repairing code. This is also an area with intensive research on the application of GenAI.

4.5.2. Key RQs

Excluding open challenges in code generation or quality assurance, which have already discussed in previous sections, we present below RQs that are specific to software maintenance tasks, i.e., code translation, refactoring, and code repair:

1. How to improve code correctness in code translation tasks?
2. How can GenAI be leveraged to migrate an entire codebase to another language or framework?
3. How can GenAI be leveraged to refactor a monolithic system into its microservices counterpart?
4. How can GenAI support automatic program repair?

5. How to integrate multiple LLMs or combine LLMs with specialized traditional ML models to leverage the collective strengths for automatic program repair?
6. How can GenAI be leveraged to automate code smells and anomaly detection, facilitating the early identification and resolution of issues in software maintenance?
7. How can GenAI be leveraged to automate bug triage?
8. How can GenAI be leveraged to automate clone detection?

4.5.3. *State-of-the-art*

The past year has witnessed significant transformations in software maintenance practice, largely catalyzed by the introduction of GitHub Copilot powered by OpenAI Codex. Leveraging LLMs can assist software maintainers in a wide range of activities, including code summarization, code translation, code refactoring, code repair, and more.

Code summarization is the task of writing natural language descriptions of source code. Taking into account that according to several studies [126, 125, 124], from 40% to 60% of the software maintenance effort is spent trying to understand the system, code summarization stands out as a pivotal technique for enhancing program comprehension and facilitating automatic documentation generation. It effectively addresses the frequent disconnect between the evolution of code and its accompanying documentation. Noever and Williams [127] demonstrated the value of AI-driven code assistants in simplifying the explanation or overall functionality of code that has shaped modern technology but lacked explanatory commentary. On a related note, Ahmed and Devanbu [128] investigated the effectiveness of few-shot training with ten samples for the code summarization task and found that the Codex model can significantly outperform fine-tuned models. Apart from writing new comments, automated code summarization could potentially help update misaligned and outdated comments, which are very common in SE projects[128].

Code translation involves the intricate process of converting code between different programming languages while preserving its functionality and logic. This task holds substantial practical value, especially in scenarios like migrating legacy code to modern technologies. For instance, billions of lines written in Cobol and other obsolete languages are still in active use, yet it is increasingly challenging to find and recruit COBOL programmers to

maintain and update these systems. To bridge the gap between the available knowledge of individuals and the expertise needed to effectively address legacy problems, IBM recently introduced Watsonx Code Assistant for IBM Z mainframes. This tool helps enable faster translation of COBOL to Java and is powered by a code-generating model called CodeNet, which can understand not only COBOL and Java but also approximately 115 different programming languages.

Code refactoring is the process of restructuring and improving the internal structure of existing source code without changing its external behavior. Madaan et al. [129] explored the application of LLMs in suggesting performance-enhancing code edits. They observed that both Codex and CodeGen can generate such edits, resulting in speedups of more than $2.5\times$ over 25% of the investigated programs in both C++ and Python. Notably, these improvements persisted even after compiling the C++ programs using the O3 optimization level. On a related note, Poldrack et al. showed that GPT-4 refactoring of existing code can significantly improve code quality according to complexity metrics [1].

Code repair refers to the identification and fixing of code errors that may cause program crashes, performance bottlenecks, or security vulnerabilities. Accelerating this process using LLMs is crucial, as developers invest a significant amount of engineering time and effort in identifying and resolving bugs that could otherwise be devoted to the creative process of software development. Automated program repair (APR) has been a topic of much interest for over a decade in the SE research community. Zhang et al. proposed a pre-trained model-based technique to automate the program repair [130]. Xia et al. [131] designed three different repair settings to evaluate the different ways in which state-of-the-art LLMs can be used to generate patches: 1) generate the entire patch function, 2) fill in a chunk of code given the prefix and suffix 3) output a single line fix. The findings suggest that directly applying state-of-the-art LLMs can substantially outperform all existing APR techniques. In contrast, Pearce et al. [132] examined zero-shot vulnerability repair using LLMs (including Codex) and found that off-the-shelf models struggle to generate plausible security fixes in real-world scenarios. In a different context, Asare et al. [133] discovered that Copilot has the potential to aid developers in identifying potential security vulnerabilities. However, it falls short when it comes to the task of actually fixing these vulnerabilities. Lastly, Mastropaolo et al. [96] fine-tuned the T5 model by reusing datasets used in four previous works that used deep learning techniques to

(i) fix bugs, (ii) inject code mutants, (iii) generate assert statements, and (iv) generate code comments. They compared the performance of this model with the results reported in the four original papers proposing deep learning-based solutions for those four tasks. The findings suggest that the T5 model achieved performance improvements over the four baselines.

4.5.4. *Challenges and Limitations*

As detailed in the preceding subsections, GenAI models demonstrate proficiency in maintenance tasks that require an understanding of syntax, such as code summarization and code repair. However, they encounter challenges when it comes to grasping code semantics [122, 134]. Consequently, the performance of GenAI is still limited when it comes to several realistic scenarios [85].

Moreover, applying available GenAI tools to code translation tasks also faces challenges and limitations. Primarily, the amount of source code that LLMs can consider when generating a response is constrained by the size of the context window. To illustrate, ChatGPT Plus imposes a token limitation of 8,000 characters, equivalent to roughly 100 lines of code. Consequently, when dealing with extensive codebases, a straightforward approach of inputting the entire source code at once becomes unfeasible. For instance, in the case of GPT Migrate¹¹, code migration is a solvable problem if the entire codebase is small enough to fit in the context window. However, addressing larger codebases remains an unresolved issue. As this restriction on code length is inherent in the transformer architecture, overcoming this limitation would necessitate a significant breakthrough in the field. Besides, acquiring a large amount of high-quality source code and target code pairs is challenging, which may limit the model’s learning and generalization capabilities. Finally, as even minor translation errors can render the generated code non-functional, code correctness and precision are of paramount importance, posing challenges for LLMs.

4.5.5. *Future Prospects*

At the current stage, LLMs can serve as useful assistants to software maintainers, but they are very far from replacing completely humans [122]. Nonetheless, LLMs are continuously advancing, and they still hold significant

¹¹<https://github.com/0xpayne/gpt-migrate>

untapped potential. One promising avenue is the concept of Collaborative LLMs [134]. This approach involves the integration of multiple LLMs or their combination with specialized ML models to minimize the necessity for human intervention. In this setup, multiple LLMs function as distinct "experts", each handling a specific subtask within a complex task [82]. By harnessing the collective strengths of diverse models, more precise and efficient outcomes can be achieved [134]. Furthermore, regarding the enhancement of correctness and performance in LLMs for vulnerability detection, future research should focus on combining LLMs with formal verification techniques, incorporating previously related edits, or implementing self-debugging algorithms [122].

4.6. GenAI in Software Processes and Tools

4.6.1. Historical Context.

Software processes are essential components of software development, helping developers create, maintain, and deploy software applications efficiently and effectively. Industry continues to look for novel ways of improving SE processes. So far, common software processes include Waterfall, Agile, Continuous Software Engineering and Lean Software Development [135, 136, 137]. The implementation of software processes often goes hand in hand with the adoption of various tools. For instance, Agile projects utilize Kanban-based management tools, and Continuous Software Engineering with Continuous to automate the build, testing, and deployment of software, enabling rapid and reliable releases. As the latest movement, ML/AI for SE and specifically GenAI is receiving widespread interest in this regard. While early pioneers have been investigating and employing ML in SE even before the popularity of systems like ChatGPT, the integration of GenAI into SE practices has emerged as a critical industry priority. Although individual developers have started embracing these technologies for personal assistance[138], the endorsement and implementation of GenAI tools at the company level by management in many organizations are lagging.

4.6.2. Key RQs.

In this regard, the main question is *what* to do with GenAI, and *how* to do so effectively while considering the current working processes, practices and environments. For example, a practitioner looking into Copilot could ask which tasks are the tool suited for and what are the good and best practices for utilizing them for said tasks. Given the versatility of GenAI, the potential

applications of these tools in SE are still being explored. To this end, as GenAI tools are used to support and automate tasks by different personnel, the question of how they impact productivity becomes increasingly relevant from a business point of view. In the long run, the question of how these tools will transform SE processes and what types of new processes are born due to their use is important. We highlight specific RQs related to the SE process as below:

1. What strategies can be developed to enable GenAI models to support continuous integration and deployment (CI/CD) practices?
2. How can different GenAI tools be incorporated into software development environments?
3. To what degree do GenAI tools improve developer productivity?
4. How can GenAI be utilized to aid in the automation of code commit, pull, and push requests?
5. What approaches can be used to enhance the integration of GenAI into development environments, providing real-time intelligent assistance and feedback during code editing and development activities?
6. How can GenAI support the automation of build and deployment processes?
7. What is the impact of GenAI tools on developer productivity in various software development tasks and across different levels of developer seniority?
8. How can GenAI contribute to Citizen development?
9. How will GenAI tools transform fast-faced software development processes, i.e. Agile, Continuous Development and Lean Development?

4.6.3. State-of-the-art.

There currently exist a few studies focusing on GenAI and software processes. The tertiary study by Kotti et al. [18] provides an overview of AI use in SE overall, reconsidering deficient SE methods and processes. Weisz et al. [139] and Whites [46] explore the applications of GenAI in programming-related tasks, while in another work, Whites and co-authors also discuss prompt patterns for SE-related prompts [73]. Ross et al. proposed a programmer assistant based on LLMs, and explored the utility of conversational interactions grounded in code [140]. The impacts of the utilization of GenAI tools have also been discussed, primarily from the point of view of productivity [100]. While some existing papers speculate on how SE might be

transformed by (Gen)AI going forward, this remains on the level of speculation, although Bird et al. [102] provide some insights into how the use of GitHub Copilot changes the way software developers work.

Software processes are more often presented as a part of study contexts. Petrovic et al. presented a use case where ChatGPT can be used for runtime DevSecOps scenarios [141]. In a game development context, Lanzi et al. proposed a collaborative design framework that combines interactive evolution and large language models to simulate the typical human design process [142]. Ye et al. investigated the impact of ChatGPT on trust in a human-robot collaboration process [143].

4.6.4. Challenges and Limitations.

While many companies are interested in the potential prospect of automation and the resulting productivity and financial gains, many existing papers highlight the need for human oversight in utilizing GenAI in SE processes due to the potential errors GenAI tools make (e.g., programming [144]). In this regard, Moradi et al. [78] argue that tools such as GitHub Copilot can be assets for senior developers as supporting tools but may prove challenging to more novice developers who may struggle to evaluate the correctness of the code produced by the AI. Becoming overconfident in GenAI tools may result in a lack of criticism towards the code produced [79]. Overall, good practices in utilizing GenAI in the SE process are still missing, and only a number of early studies have begun to inspect the practical applications of GenAI in the SE process. Though the promise of code generation with GenAI is now largely acknowledged and, to a large extent, demonstrated in practice as far as market viability is concerned by the explosive growth of the GitHub Copilot service, the utilization of GenAI for other SE processes is far less explored, and even the impact of GenAI on code generation remains poorly understood (e.g., productivity in different contexts).

4.6.5. Future Prospects.

GenAI tools, alongside AI in general, have the potential to transform SE processes as we know them today. It is argued that, in the near future, the work of developers will shift from writing code to reviewing code written by AI assistants [102]. Carleton et al. [9] argue that the key is "shifting the attention of humans to the conceptual tasks that computers are not good at (such as deciding what we want to use an aircraft for) and eliminating human error from tasks where computers can help (such as understanding the

interactions among the aircraft’s many systems)”. Such larger, fundamental shifts in organizing SE processes are arguably at the center of the future prospects for the SE process research area. For example, while the changing role of developers is acknowledged, with AI tools making it possible for developers to automate menial tasks in order to focus on tasks that require more expertise, what exactly these tasks might be in the future remains an open question.

4.7. GenAI in Engineering Management

4.7.1. Historical Context

- Engineering management is the SE branch that deals with managing principles and practices concerning the engineering field. The critical aspects of this field include project management [145], human resources [146], team, leadership [147], technical expertise, budgeting, resource allocation, risk management [148], communication, quality assurance, continuous improvement, ethics, and professionalism [149]. A project undertaken under engineering management ensures that it is completed successfully, on time, within budget, and with high-quality standards [150]. Crucial is to manage the project under several laid out regulations [149]. Tambe et al., in Human resource management and AI, identified four types of challenges. First, complex phenomena surround the human resource. Second, the chain that bounds the small data sets, i.e., the people working in an organization, is minor in comparison to the purchases made by the customer. Third, ethical obligations are involved with legal constraints. Fourth, the response or behaviors of employees towards the management [146]. Wang et al. proposed a systematic roadmap to navigate the next step of AI in product life cycle management. The study further mentions the advantages and challenges of applying AI in product cycle management. [151].

4.7.2. Key RQs

- We consider the following RQs that present some open challenges in managing software development projects:

1. What are the implications for team sizes and dynamics of the use of GenAI in software development?
2. Can a trained AI tool speed up the enrolment process of a new employee?

3. To what extent should experience developers be retrained to adapt to GenAI use in their work?
4. How can organizations foster a culture that promotes the symbiotic collaboration between software developers and GenAI systems, creating an environment that encourages trust, learning, and innovation?
5. What could be an efficient human-ai collaboration workflow?
6. Do different SE roles feel a stigma about using AI assistants to help them with their jobs?
7. What is the economic impact of incorporating GenAI in software development processes, considering factors such as cost savings, productivity gains, and overall return on investment?

4.7.3. State-of-the-art

For project management in general, Holzmann et al. conducted a Delphi study on 52 project managers to portray the future AI applications in project management [152]. The author presented the list of most and least important project management areas where GenAI can be adopted. Prifti et al. identified the pain points or problems in project management. Then, the paper proposes how AI can play a role and help optimize weak points. AI's assistance helps project managers become more productive and skilled in organizations [153].

In software project management, AI-based approaches can be useful for various project estimation and allocation tasks. Dam et al. explored the effort estimation problems in an Agile context [154]. Elmousalami analyzed and compared the 20 AI techniques for a project cost prediction [155]. Fu et al. proposed a GPT-based Agile Story Point Estimation approach, highlighting a practical need for explanation-based estimation models [156]. Alhamed evaluated the Random Forest and BERT model in software ask effort estimation. [157]. The authors suggested that BERT shows marginally improved performance than other approaches.

To contribute to the discussion about GenAI in an industrial context, Chu explored the impact of ChatGPT on organizational performance (productivity), reporting various use contexts but no notable productivity increases [158], although without a specific focus on SE-related work. Ebert & Louridas [159] reported industrial, empirical results from a case study, discussing the overall promise of GenAI for software practitioners and the current state of the field, and highlighting potential application contexts in

Most important PM functions for GenAI adoption
<ul style="list-style-type: none"> • Create project schedule • Analyze implications of missing deadlines • Create WBS and tasks list • Create project budget • Identify scope creep and deviation • Produce dynamic risk map • Extract deliverables • Update project progress and schedule • Prioritize tasks • Allocate team

Table 3: A list of relevant and important PM tasks can be supported by AI tools [152]

SE (*media content creation and improvement, generative design, code generation, test case generation from requirements, re-establishing traceability, explaining code, refactoring of legacy code, software maintenance with augmented guidance, and improving existing code, among others*).

4.7.4. Challenges and Limitations

In this SE area, the lack of industrial studies, and especially the lack of studies with an organizational focus, presents as one of the notable gaps. Pan et al. presented a critical review of AI-generated future trends in construction engineering and management. The study outlines six crucial concerns regarding AI in construction engineering, which are (1) knowledge representation, (2), information fusion, (3) computer vision, (4) natural language processing, (5) process mining, and (6) intelligence optimization [160]. Another study described the future challenges of adopting AI in the construction industry [161], presenting several challenges. They are the cultural issues and explainable AI, the security concerns, the talent shortage of AI engineers, the high cost in the construction industry, ethics, governance, and finally, the computing power and internet connectivity that lead to construction activities. There are several challenges concerning software project management, too, such as needing more understanding concerning the software development team and awareness of the project. The project context could be complex, with limited knowledge available that might create developers in the group with biased decisions. In other words, biased data created could lead to subjective decisions that result in less effective software projects [162].

4.7.5. Future Prospects

AI is regarded as an originator, coordinator, and promoter of innovation management. At the company level, AI applications can be used for opportunities like searching for information, generating ideas, and value creation. In terms of organization design, AI could be beneficial in looking into the structure of the firm, customer engagement, and human decision-making processes. AI can be useful for generating information to data access that could help at the startup and SME levels [163]. In software project management, AI can be very handfull in assisting the organization of resources and allotting, managing the risks involved in the projects, automation of the task allocation, estimating the cost of the project and its related requirements and planning the agile sprints [154]. Monitoring the performance, capturing the

problems inside the project, providing support suggestions, and documenting the project could make some prospects replaceable by AI [164].

4.8. GenAI in Professional Competencies

4.8.1. Historical context.

There are skill shortages and talent gaps [165, 166, 167] in the software industry: this situation has been going on for a while and companies struggle to find qualified employees [168, 169]. On top of that, there is a gender gap in technology-related study fields [168, 170] and a soft skills gap [171, 165] that make it even harder to find professionals and have balanced teams able to detect and correct unfair biases, e.g., caused by the selection of data sources [172]. The question then is how to offload repetitive tasks from existing developers by smoothly integrating GenAI into their workspace. On the one hand, as happened with previous technologies empowering developers, integrating GenAI into the workspace and ensuring that developers smoothly work alongside GenAI tools will probably require training [167]. However, as seen in previous research, some users might already have an intuition on how to work with them, maybe depending on their seniority level or background [166, 167]. On the other hand, GenAI could also be part of the training of software developers of the future, partially addressing the aforementioned skill shortages, as discussed in Section 4.9.

4.8.2. Key RQs.

Not only academic research teams are studying how to effectively integrate GenAI within Agile teams to increase their productivity, also companies are actively exploring how these technologies will affect the work dynamics of their employees and, more specifically, of developers. Future research should explore the integration of GenAI tools in professional practices, covering productivity impacts, necessary skills and training, addressing skill shortages, GenAI as an advisor, and perceptions of integration strategies in Agile software development. These high-level concerns are addressed by the five suggested RQs listed below:

1. How can GenAI address the skill shortages and talent gaps in the software industry, and how would different strategies affect the desired profile for future software developers?
2. How would various strategies for GenAI integration within Agile software development, such as in up-skilling and re-skilling professionals,

offloading repetitive tasks, or human resource management, be perceived?

3. What is the impact of GenAI tools on productivity across various software development tasks, with respect to developer seniority and at the organizational level at which the GenAI interventions take place (i.e., team or team member)?
4. How can GenAI be effectively used as an advisor to enhance the productivity and skills of software professionals, and what are the key challenges and opportunities in designing training programs for this?
5. What skills and training are necessary for software engineers to generate quality outputs using GenAI and effectively work alongside GenAI systems? Does the level of SE training correlate with these goals?

These RQs reflect the main lines of research from industry and academia in understanding how GenAI tools could mitigate the current issue of finding software professionals and either offload or up-skill existing developers to strengthen their teams.

4.8.3. State-of-the-art

As previously stated, both academic and industrial researchers are actively exploring how GenAI technologies will affect the work processes and dynamics. An increasingly adopted tool for such explorations is GitHub Copilot [77]. Copilot generates pieces of code given prompts in the form of comments or function signatures thanks to its training with open-source projects hosted by the GitHub platform. Other platforms include local versions of GPT-models [173] or BARD [174], both based in LLMs. For these tools, however, it is unclear whether their usage violates the licenses and internal best security practices. As a result, informed human oversight will still be needed to ensure the trustworthiness of the generations [175].

4.8.4. Challenges and limitations.

Software development companies are often cautious about granting external access to their systems due to strict policies safeguarding proprietary information and sensitive data. Convincing companies to participate in research on GenAI interactions can be challenging, even though demonstrating the mutual benefits of testing the integration of GenAI tools in their unique working environment can increase their willingness to collaborate. As in any other research endeavour involving human participants, obtaining informed

consent and maintaining confidentiality and anonymity are crucial. However, in this case, research results may offer insights into the different productivity or work dynamics of teams and team members. Therefore, researchers must ensure that their findings are put into context and that the validity of their conclusions is clearly stated. This precautionary approach is essential to prevent companies from making drastic decisions based on potential misinterpretations of the research results.

4.8.5. Future prospects.

In conclusion, addressing these RQs and carefully applying the results to various work environments and domains can serve as a valuable guide for the sensible integration of GenAI tools within Agile software development. This integration should consider the progressive adaptation of work dynamics and their adoption by the different stakeholders. The answers to these questions will, on the one hand, prepare companies and industrial players for the new technological setup where software developers can be offloaded of repetitive tasks. On the other hand, this knowledge will play a crucial role in shaping the integration of GenAI into educational and re-skilling activities for future software developers, as discussed in Section 4.9, indirectly contributing to mitigate the present talent gap [165, 167].

4.9. GenAI in Software Engineering Education

4.9.1. Historical Context.

The development of engineering education has been inextricably linked to the technological development of the era. Technological adoption for better education is always an important topic in higher education. SE research has extensively looked at the use of chatbots to support students outside the classroom. These chatbots can provide automated assistance to students, helping them with coding and problem-solving tasks [7], providing explanations and examples, as well as helping students identify errors in their solutions [176, 177, 178, 179, 180].

Another popular SE education paradigm is an automated assessment of exercises. Automated assessment techniques can help provide students with immediate feedback on their work, allowing them to quickly identify and correct errors [181]. This approach can also facilitate self-assessment and self-examination among students, enabling them to explore various solutions and identify areas where they require further improvement [182]. Another active research area that is gaining attention is how learning can be individualized.

Current AI approaches focus on finding questions of similar difficulty to a given question [183].

4.9.2. Key RQs

GenAI has the potential to revolutionize the existing education landscape by advancing the three above trends. For future research actions, there are several interesting themes that can be framed as RQs, as listed below:

1. How can GenAI be effectively integrated into SE education curricula to enhance student learning outcomes and develop future-ready skills?
2. In a classroom environment with a “virtual” assistant with the capacity of i.e. ChatGPT, how can it be set up to effectively achieve the learning objectives?
3. How can GenAI be used to personalize and adapt SE education to cater to diverse student needs, learning styles, and skill levels?
4. How does the integration of GenAI in SE education foster critical thinking, problem-solving skills, and computational creativity among students?
5. What assessment methods and tools can be developed to evaluate students’ understanding, competency, and proficiency in GenAI concepts and applications?
6. What are the long-term impacts of including GenAI in SE education, such as its influence on students’ career paths, job prospects, and contributions to the technology industry?
7. In what activities the use of AI can harm the learning outcome that the participants have?
8. What are the ethical considerations and implications of teaching GenAI in SE education, and how can educators promote responsible AI usage and ethical decision-making among students?
9. What are the challenges and opportunities in training IT educators to effectively incorporate GenAI concepts and tools into their teaching practices?

4.9.3. State-of-the-art

We found a large number of studies discussing the potential of GenAI in education in general and in SE education, in particular, [184, 185, 186, 187, 188, 189]. Bull et al. interviewed industry professionals to understand current practices, implying a visionary future of software development education

[185]. Jalil et al. explored opportunities for (and issues with) ChatGPT in software testing education [188].

There also emerges empirical evaluation of GenAI in different teaching and learning activities. Savelka et al. analyzed the effectiveness of three models in answering multiple-choice questions from introductory and intermediate programming courses at the post-secondary level [190]. A large number of existing papers discuss the potential challenges GenAI presents for educators and institutions, which is also a recurring topic in these SE papers. However, these papers also discuss the potential benefits of GenAI for SE education. Yilmaz et al. conducted experiments with their students and found that ChatGPT could enhance student programming self-efficacy and motivation, however, it is important to provide students with prompt writing skills [191]. Philbin explored the implications of GenAI tools for computing education, with a particular emphasis on novice text-based programmers and their ability to learn conceptual knowledge in computer science. The author presented many positive angles on GenAI adoption and suggested the integration of basic AI literacy in curriculum content to better prepare young people for work [192]

4.9.4. Challenges and Limitations

While GenAI tools like ChatGPT offer a revolutionary approach to facilitating SE education, they are not without limitations. A recent paper presented the perceived weaknesses of ChatGPT in nine SE courses. The authors presented the weaknesses in three categories: theory courses, programming courses, and project-based courses.

In a theory course, a common challenge of ChatGPT is that its responses are limited to the information provided in the question. This means that if a question is poorly formulated or lacks relevant details, the response from ChatGPT may not be satisfactory. To use ChatGPT effectively, students need to be prepared with critical and innovative thinking, which does not yet exist in any college's curriculums. The problem of the non-deterministic nature of AI is particularly significant in an educational context, as different answers can be generated for the same question, which could lead to confusion for students. Besides, ChatGPT has limited domain knowledge, so questions that require a lot of context or background information may not receive meaningful answers without sufficient input from the user.

Regarding programming courses, while GenAI tools can provide information and guidance on programming concepts, students cannot develop their

practical skills solely by using ChatGPT. This applies to complex programming concepts, intricate problems, especially those related to integrating libraries or, specific system configurations, or large software systems. AI tools sometimes offer deprecated solutions or code snippets, meaning they are outdated or no longer recommended for use. Using deprecated solutions can lead to inefficiencies or conflicts in modern coding environments. Moreover, with lectures with a lot of visual representations, ChatGPT relies on natural language processing and may struggle to interpret visual information. For example, in a lecture about 3D modeling, ChatGPT may not be able to explain a particular aspect of the modeling process that is best understood through visual demonstration. While ChatGPT can provide information on general programming concepts, it may not be able to provide insight into specific practices used in a particular industry. For example, a student studying game development may need to learn about specific optimization techniques used in the gaming industry, which ChatGPT may not be familiar with. Furthermore, as technology and programming languages evolve rapidly, AI's knowledge might lag behind the most recent advancements or best practices. Some students pointed out that e.g. ChatGPT lacked information about the latest developments, affecting its ability to provide the best or most relevant answers. Furthermore, AI models don't possess an inherent understanding of context. There were situations where the tool couldn't recognize missing methods in a Repository class or provide solutions based on a broader project context, which a human tutor might be able to grasp.

Regarding project-based learning, where project contexts can change frequently, it is necessary to update the context information for the tool to catch up with project progress. For example, if a project in a software engineering class changes its requirements, students may need to make changes to their code. ChatGPT may not be able to keep up with these changes and may not be able to provide relevant advice to students. This is also because real-world settings can not be fully described as input for the tool. ChatGPT is based on pre-existing data and may not be able to provide practical advice for unique situations that occur in real-world settings.

4.9.5. Future Prospects

In conclusion, GenAI tools, like ChatGPT, have shown significant promise in SE education. In the future, We foresee a comprehensive integration of GenAI tools at different levels for both educators and learners. Future research will address today's gap to visualize this scenario:

- **The integration of GenAI into SE teaching content and approaches.** There is a pressing need for educational institutions to adapt their curriculum and teaching methodologies to incorporate GenAI concepts, tools, and practices. Concerning curriculum development, it becomes essential to determine which GenAI-related knowledge, such as prompt engineering, AIOps, and AI-human interaction, should be integrated into existing lectures on core SE topics like requirements engineering, software design, software testing, and quality assurance. On the teaching approach, how should such integration be done? GenAI tools like CoPilot, and ChatGPT also present a great potential to assist both educators and learners. There is a need to develop methodologies that help educators and AI developers work collaboratively to ensure that AI recommendations and adaptations align with established learning outcomes and instructional strategies. Besides, while AI can leverage large volumes of data to personalize learning experiences, there is a lack of standardized methodologies for collecting, processing and utilizing educational data ethically and effectively.
- **A new assessment approach for student-GenAI collaboration** A significant knowledge gap exists concerning the development and implementation of a novel assessment approach tailored to evaluate the efficacy and outcomes of student-GenAI collaboration in educational settings. We can not assess the student project delivery without considering the level of CoPilot involved in their project. Research is needed to create fair and reliable assessment techniques that differentiate between student-generated and AI-generated work. Project requirements and learning objectives might need to be updated as well to ensure that evaluations accurately reflect students' abilities and knowledge
- **An evaluation of the impact of AI-enabled curriculum and teaching approaches on learning outcome and student engagement** It is necessary to delve into empirical studies to measure and evaluate the impact of GenAI-related content and teaching approaches on student learning outcomes and engagement. Research in this area can involve conducting controlled experiments and collecting data to quantify the effectiveness of AI-assisted approaches compared to traditional methods. Assessment needs an ethical guideline focusing on how GenAI is used in SE courses.

4.10. GenAI in Macro aspects

4.10.1. Historical context

The use of GenAI for SE brings implications and opportunities to other spheres beyond the technical aspects, such as market, business, organizations and society. This phenomenon can be observed with other technologies. For example, the advent of cloud computing, in which computing resources are hired on demand without the need for large upfront investments in infrastructure, allowed the cheap experimentation of new digital products [193], paving the way for methodologies for the development of startups focused on quick feedback from customers, such as Customer Development [194] and Lean Startup [195]. A more recent example is the emergence of more powerful AI algorithms and their reliance on data, leading to a growing research interest in the ethical aspects of this use [196], i.e., AI ethics.

4.10.2. Key RQs

We present the key RQs identified regarding macro aspects as listed below:

1. How can GenAI be employed to support novel and innovative software-intensive business models?
2. How can the adoption of GenAI in software development affect the competitiveness and market positioning of software-intensive companies, and what strategies can be employed to leverage AI for business growth?
3. What are the potential barriers and challenges to the adoption of GenAI in software-intensive businesses?
4. How does the integration of GenAI in software-intensive businesses affect software licensing models, and intellectual property rights?
5. How can software businesses effectively navigate the ethical considerations and societal impacts of using GenAI?
6. How does the adoption of GenAI impact the sustainability of software businesses?

The first aspect included in this section regards the business of software. Any new technology brings several economic opportunities usually exploited by new companies. The application of GenAI to software development could bring several innovation opportunities for the players involved. A better understanding of this process, including a clear view of its associated challenges, would help software startups and established companies create and capture

value for their customers by developing novel or adapting existing business models effectively. Still, regarding economic aspects, it remains unclear how the fact of AI creating intellectual property (IP) could change the management of this IP, including protection, use, and monetization. The second aspect regards the implications of the use of GenAI in software development to the environment and society. For example, GenAI adoption might reduce the need for human software developers, leading to a large contingent of job losses and the consequent need for retraining these people. Another issue might be the environmental impact of GenAI given, for example, the amount of resources needed to train the models.

4.10.3. State-of-the-art

Moradini conducted a narrative review to analyze research and practice on the impact of AI on human skills in organizations [197]. The authors suggested the identification of needed skills for the adoption of AI and provided training and development opportunities to ensure their workers are prepared for the changing labor market. Kanbach et al. [198] performed a business model innovation (BMI) analysis on the implication of GenAI for businesses. The authors identified some propositions based on the impact on innovation activities, work environment, and information infrastructure. In particular, they discussed the SE context identifying possibilities for value creation innovation, e.g., automatic code generation, new proposition innovations, e.g., AI-powered software development platforms, and value capture innovations, such as improving operational efficiency by, for example, task automatization. On the other hand, there have been some calls for action regarding the regulation of GenAI, for example, in the healthcare sector [199].

4.10.4. Challenges and Limitations

As essentially interdisciplinary aspects, the research on these topics will have to converge to other fields, such as economics, psychology, philosophy, sociology, and anthropology. For example, concerns regarding intellectual property involve not only economic but also more fundamental aspects, such as what a person is, that are the study field of philosophers. These aspects are also associated with national and supranational politics and economics. Different countries might decide to create specific legislation or public policies depending on their political and economic tradition.

4.10.5. Future Prospects

GenAI is an emerging technology that will develop even further in the following years. As it becomes increasingly capable of performing SE-related tasks, its implications on the macro aspects discussed above will grow. GenAI will be increasingly employed as part of software-intensive business models or even to suggest modifications to these business models. This process will represent a challenge for consolidated companies but also an opportunity for new incumbents. Current market leaders will need novel strategies to cope with these challenges and innovate, and novel software-intensive business models will emerge. In the following decades, we will probably see the downfall of current juggernauts and the emergence of new ones. Not only companies but also individuals will be affected, increasingly being replaced by AI. In this regard, we will probably watch a change in labour force, including professions related to software development, towards new careers, many of these initiated by GenAI, in a similar movement that developed countries experienced after the deindustrialization and the transfer of labour force to the services segment. Finally, we will probably see in the following years the emergence of new legislation regulating the use of GenAI and the products created by it. This phenomenon could lead to tensions among different actors in society, such as the strike of writers in Hollywood that paralyzed the production of several movies and TV series, motivated, among other things, the use of GenAI for script generation. Similarly, it could lead to international tensions regarding different legislation in different countries or regions or the influence of one country that detains the technology over others consuming it.

4.11. Fundamental concerns of GenAI

GenAI models, particularly those based on deep learning architectures such as the GPT series, have shown remarkable capabilities in various SE tasks ranging from code generation to bug detection. However, their application in SE also brings a set of fundamental concerns. We summarized the concerns in the following RQs and elaborated on them further in this section:

1. How all previous RQs can be addressed in an industry-level evaluation?
2. How reliability and correctness of GenAI can be ensured for professional uses?
3. How can GenAI output be explained and systematically integrated into current decision-making mechanisms?

4. How sustainability concerns can be addressed for the application of GenAI?

Industry-level evaluation of GenAI. LLMs are seldom effective when used in a small task but can be highly effective as part of an overall software engineering process. The industry-level evaluation of GenAI technologies would be the next step to bridge the gap between small-sized empirical studies and real-world applications, ensuring that the promises made in research labs align with practical industry needs and challenges. Industry evaluations might reveal the tangible impacts, such as feasibility, usability, scalability, and cost-value tradeoff of adopting GenAI. Scalability is a critical factor for industry evaluation. Can the AI system perform well at scale, handling large volumes of data and complex tasks? Usability, user satisfaction, and user engagement are key aspects of evaluation. Moreover, research is desirable for understanding workflows, processes, and coordination paradigms so that GenAI can safely, efficiently and effectively reside. Moreover, in applications where AI interacts with users, the user experience is vital.

Reliability and correctness concern The reliability and correctness of the generated code are pivotal. Unlike creative content generation, where occasional mistakes might be acceptable, in SE, even minor errors can have catastrophic consequences [200]. However, there are challenges when rolling out improvements: improving a model’s performance on certain tasks can unexpectedly affect its behavior in other tasks (e.g., fine-tuning additional data might enhance performance in one area but degrade it in another). The problem of hallucination has already been widely studied [8, 201, 202, 203, 204]. It will continue to be a topic of great interest, both within the software engineering community and in the wider computer science community. Moreover, the performance and behavior of both GPT-3.5 and GPT-4 vary greatly over time [205]. For instance, GPT-4’s March 2023 version had a better performance at identifying prime numbers compared to its June 2023 version, a relatively short period. This emphasizes the importance of continuous evaluation, especially given the lack of transparency regarding how models like ChatGPT are updated.

Data availability AI model relies heavily on a large amount of data for training and fine-tuning, posing challenges for private companies to adapt existing AI models to their work. The quality, diversity, and quantity of data directly affect the performance and generalizability of the models. Domain-specific data required for fine-tuning can be a bottleneck. Moreover, privacy

and security are major concerns for companies and organizations adopting GenAI. Company data might be confidential, sensitive and personal. Concerning the adoption of GenAI in private companies can include concerns about data breaches, unauthorized access to confidential data, and the use of data without permission [206, 207, 203]. Regular audits of the data privacy and security measures will need to be in place to identify and address any potential vulnerabilities or areas for improvement.

Explainability concern. The “black-box” nature of these models poses interpretability challenges, making it difficult to reason about their decision-making process or to debug when things go awry [208, 209]. Transparency in decision-making is crucial in SE, where understanding the rationale behind every line of code is often necessary. Other ethical concerns also arise when discussing integrating GenAI tools in real-world environments. For instance, the dependence on training data means that these models can inadvertently introduce or perpetuate biases in the datasets they are trained on, leading to unfair or discriminatory software applications [172]. These biases can be perpetuated if appropriate human oversight mechanisms, which require transparency, are not set in place. For instance, mechanisms to avoid automation bias are largely needed, especially for LLMs that usually make confident statements about their decisions. This could lead, in the future, as stated by the participants in the survey conducted by Cabrero-Daniel et al., to a potential loss of human expertise as tasks become automated and GenAI tools become common-use [210], leading to a diminished ability to address or understand complex software challenges in the future manually.

Sustainability concerns. As a general concern in sustainable software engineering [211, 212, 213], the appearance of lots of LLMs daily raise a concern about their environmental impact, i.e. energy consumption [214, 215, 216]. Chiet et al. showed that ChatGPT-like services, inference dominates emissions, in one year producing 25 times the CO2 emissions of training GPT-3 [215]. LLM capabilities have been associated with their size [217], resulting in the rapid growth of model size [218]. While it has been suggested that the model performance depends not only on model size but also on the volume of training data [74], the question of a triple-bottom-line model size that at the same time achieves performance, environmental and organizational objectives, remains unclear.

5. Outlook and Conclusions

The field of GenAI in SE is an important and fast-moving SE research area, currently presenting opportunities as well as open challenges. GenAI has the power to redefine the way software is designed, developed, and maintained. The innovative capabilities of AI-driven systems promise to streamline processes, boost efficiency, and open up new avenues for creative problem-solving. However, there are also common challenges in adopting GenAI and specific challenges for SE activities, i.e., requirements engineering, software implementation, quality assurance, and SE education. Our research agenda, outlined in this paper, stands as one of the pioneering initiatives to systematically reveal the state of research as of Oct 2023. Software Engineering is a large spectrum that includes technical activities but also user-centric, process-driven and managerial aspects. A comprehensive view should consider insights and methodologies from research about software project development, software project management, professional competencies, software engineering education, and software businesses. Therefore, collaboration with experts from these diverse disciplines is essential to harness the full potential of this research area.

In light of the evolving nature of GenAI research, significant work remains to establish GenAI in SE as a mature research area. Substantial theories must be developed to provide a solid theoretical underpinning for the practical applications. Technological advancements should be validated in a realistic project context. The adoption of GenAI in SE can also be combined with other active research areas, e.g., sustainability, trustworthy systems, and education.

We acknowledge that this research agenda might still miss some important SE aspects where GenAI can be applied. RQs are presented to illustrate significant concerns in each SE area and are not meant to be exhaustive. The research agenda is open to additions of new tracks, topics, and RQs by other researchers interested in the research area. Through this collective effort and commitment from a diverse community of researchers, we can strengthen the foundations of GenAI in SE and ensure its relevance in a rapidly changing technological landscape.

Declaration of Generative AI and AI-assisted technologies in the writing process During the preparation of this work, we used ChatGPT version 3.5 and Grammarly services to edit and improve our writing, i.e. rephrasing sentences and fixing typos and language mistakes. After using this tool/service, we reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

References

- [1] R. A. Poldrack, T. Lu, G. Beguš, AI-assisted coding: Experiments with GPT-4. arXiv:2304.13187 [cs], doi:10.48550/arXiv.2304.13187.
URL <http://arxiv.org/abs/2304.13187>
- [2] P. Denny, V. Kumar, N. Giacaman, Conversing with copilot: Exploring prompt engineering for solving CS1 problems using natural language. arXiv:2210.15157 [cs], doi:10.48550/arXiv.2210.15157.
URL <http://arxiv.org/abs/2210.15157>
- [3] J.-B. Döderlein, M. Acher, D. E. Khelladi, B. Combemale, Piloting copilot and codex: Hot temperature, cold prompts, or black magic? arXiv:2210.14699 [cs], doi:10.48550/arXiv.2210.14699.
URL <http://arxiv.org/abs/2210.14699>
- [4] Y. Dong, X. Jiang, Z. Jin, G. Li, Self-collaboration code generation via ChatGPT. arXiv:2304.07590 [cs], doi:10.48550/arXiv.2304.07590.
URL <http://arxiv.org/abs/2304.07590>
- [5] S. Ouyang, J. M. Zhang, M. Harman, M. Wang, LLM is like a box of chocolates: the non-determinism of ChatGPT in code generation. arXiv:2308.02828 [cs], doi:10.48550/arXiv.2308.02828.
URL <http://arxiv.org/abs/2308.02828>
- [6] Y. Liu, G. Deng, Z. Xu, Y. Li, Y. Zheng, Y. Zhang, L. Zhao, T. Zhang, Y. Liu, Jailbreaking ChatGPT via prompt engineering: An empirical study. arXiv:2305.13860 [cs], doi:10.48550/arXiv.2305.13860.
URL <http://arxiv.org/abs/2305.13860>
- [7] W. Sun, C. Fang, Y. You, Y. Miao, Y. Liu, Y. Li, G. Deng, S. Huang, Y. Chen, Q. Zhang, H. Qian, Y. Liu, Z. Chen, Automatic code summarization via ChatGPT: How far are we? arXiv:2305.12865 [cs],

doi:10.48550/arXiv.2305.12865.
URL <http://arxiv.org/abs/2305.12865>

- [8] H. Alkaissi, S. I. McFarlane, Artificial hallucinations in ChatGPT: Implications in scientific writing 15 (2) e35179. doi:10.7759/cureus.35179. URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9939079/>
- [9] A. Carleton, F. Shull, E. Harper, Architecting the future of software engineering 55 (9) 89–93, conference Name: Computer. doi:10.1109/MC.2022.3187912. URL <https://ieeexplore.ieee.org/abstract/document/9869582>
- [10] S. Jansen, A. Finkelstein, S. Brinkkemper, A sense of community: A research agenda for software ecosystems, in: 2009 31st International Conference on Software Engineering - Companion Volume, pp. 187–190. doi:10.1109/ICSE-COMPANION.2009.5070978.
- [11] B. Sengupta, S. Chandra, V. Sinha, A research agenda for distributed software development, in: Proceedings of the 28th international conference on Software engineering, ICSE '06, Association for Computing Machinery, pp. 731–740. doi:10.1145/1134285.1134402. URL <https://dl.acm.org/doi/10.1145/1134285.1134402>
- [12] J. Bosch, H. H. Olsson, I. Crnkovic, Engineering AI systems: A research agenda, in: Artificial Intelligence Paradigms for Smart Cyber-Physical Systems, IGI Global, pp. 1–19. doi:10.4018/978-1-7998-5101-1.ch001.
- [13] I. Sriram, A. Khajeh-Hosseini, Research agenda in cloud technologies. doi:10.48550/arXiv.1001.3259. URL <http://arxiv.org/abs/1001.3259>
- [14] R. France, B. Rumpe, Model-driven development of complex software: A research roadmap. doi:10.1109/FOSE.2007.14.
- [15] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, Service-oriented computing: a research roadmap 17 (2) 223–255, publisher: World Scientific Publishing Co. doi:10.1142/S0218843008001816.
- [16] M. Barenkamp, J. Rebstadt, O. Thomas, Applications of ai in classical software engineering, AI Perspectives 2 (1) (2020) 1.

- [17] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, S. Wagner, Software engineering for AI-Based systems: A survey, *ACM Trans. Softw. Eng. Methodol.* 31 (2), <https://doi.org/10.1145/3487043> (apr 2022).
URL <https://doi.org/10.1145/3487043>
- [18] Z. Kotti, R. Galanopoulou, D. Spinellis, Machine learning for software engineering: A tertiary study 55 (12) 256:1–256:39. doi:10.1145/3572905.
URL <https://doi.org/10.1145/3572905>
- [19] I. Paik, J.-W. Wang, Improving text-to-code generation with features of code graph on gpt-2, *Electronics* 10 (21) (2021) 2706.
- [20] M. Jovanović, M. Campbell, Generative artificial intelligence: Trends and prospects 55 (10) 107–112. doi:10.1109/MC.2022.3192720.
- [21] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: *Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS’20*, Curran Associates Inc., pp. 1877–1901.
- [22] A. Radford, K. Narasimhan, Improving language understanding by generative pre-training.
- [23] B. D. Lund, T. Wang, Chatting about ChatGPT: how may AI and GPT impact academia and libraries? ahead-of-print. doi:10.1108/LHTN-01-2023-0009.
URL <https://doi.org/10.1108/LHTN-01-2023-0009>
- [24] T. Brants, A. C. Popat, P. Xu, F. J. Och, J. Dean, Large language models in machine translation (2007).
- [25] K. Heafield, I. Pouzyrevsky, J. H. Clark, P. Koehn, Scalable modified kneser-ney language model estimation, in: *Proceedings of the 51st An-*

nual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), 2013, pp. 690–696.

- [26] Y. Bengio, R. Ducharme, P. Vincent, A neural probabilistic language model, *Advances in neural information processing systems* 13 (2000).
- [27] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, S. Khudanpur, Recurrent neural network based language model., in: *Interspeech*, Vol. 2, Makuhari, 2010, pp. 1045–1048.
- [28] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, *arXiv preprint arXiv:1301.3781* (2013).
- [29] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, *Advances in neural information processing systems* 30 (2017).
- [30] J. Lee, K. Toutanova, Pre-training of deep bidirectional transformers for language understanding, *arXiv preprint arXiv:1810.04805* (2018).
- [31] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., Language models are unsupervised multitask learners, *OpenAI blog* 1 (8) (2019) 9.
- [32] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding (2019). *arXiv:1810.04805*.
- [33] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, N. Fiedel, Palm: Scaling language modeling with pathways (2022). *arXiv:2204.02311*.

- [34] V. Lialin, V. Deshpande, A. Rumshisky, Scaling down to scale up: A guide to parameter-efficient fine-tuning (2023). arXiv:2303.15647.
- [35] A. Martakis, M. Daneva, Handling requirements dependencies in agile projects: A focus group with agile software development practitioners, in: IEEE 7th International Conference on Research Challenges in Information Science (RCIS), pp. 1–11, ISSN: 2151-1357. doi:10.1109/RCIS.2013.6577679.
- [36] J. Kontio, J. Bragge, L. Lehtola, The focus group method as an empirical tool in software engineering, in: F. Shull, J. Singer, D. I. K. Sjøberg (Eds.), Guide to Advanced Empirical Software Engineering, Springer, pp. 93–116.
- [37] J. Kontio, L. Lehtola, J. Bragge, Using the focus group method in software engineering: obtaining practitioner and user experiences, in: Proceedings. 2004 International Symposium on Empirical Software Engineering, 2004. ISESE '04., pp. 271–280.
- [38] T. Dingsøy, Y. Lindsjørn, Team performance in agile development teams: Findings from 18 focus groups, in: H. Baumeister, B. Weber (Eds.), Agile Processes in Software Engineering and Extreme Programming, Lecture Notes in Business Information Processing, Springer, pp. 46–60.
- [39] H. Edmunds, Focus Group Research Handbook, 1st Edition, McGraw-Hill.
- [40] M. Sosnowski, M. Bereza, Y. Y. Ng, Business-oriented approach to requirements elicitation in a scrum project, in: A. Przybyłek, J. Miler, A. Poth, A. Riel (Eds.), Lean and Agile Software Development, Springer International Publishing, Cham, 2021, pp. 185–191.
- [41] A. Przybyłek, M. Zakrzewski, Adopting collaborative games into agile requirements engineering, in: Proceedings of the 13th International Conference on Evaluation of Novel Approaches to Software Engineering, ENASE 2018, SCITEPRESS, 2018, p. 54–64. doi:10.5220/0006681900540064.

- [42] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, S. Shamshirband, A systematic literature review on agile requirements engineering practices and challenges, *Computers in Human Behavior* 51 (2015) 915–929. doi:10.1016/j.chb.2014.10.046.
- [43] B. Ramesh, L. Cao, R. Baskerville, Agile requirements engineering practices and challenges: an empirical study, *Information Systems Journal* 20 (5) (2010) 449–480.
- [44] K. Ahmad, M. Abdelrazek, C. Arora, A. A. Baniya, M. Bano, J. Grundy, Requirements engineering framework for human-centered artificial intelligence software systems, *Applied Soft Computing* 143 (2023) 110455.
- [45] K. Ahmad, M. Abdelrazek, C. Arora, M. Bano, J. Grundy, Requirements engineering for artificial intelligence systems: A systematic mapping study, *Information and Software Technology* (2023) 107176.
- [46] J. White, S. Hays, Q. Fu, J. Spencer-Smith, D. C. Schmidt, ChatGPT prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. arXiv:2303.07839 [cs], doi:10.48550/arXiv.2303.07839.
URL <http://arxiv.org/abs/2303.07839>
- [47] K. Ronanki, C. Berger, J. Horkoff, Investigating chatgpt’s potential to assist in requirements elicitation processes (2023) 354–361.
- [48] Q. Zhang, M. Chen, A. Bukharin, P. He, Y. Cheng, W. Chen, T. Zhao, Adaptive budget allocation for parameter-efficient fine-tuning (2023). arXiv:2303.10512.
- [49] S. Arulmohan, M.-J. Meurs, S. Mosser, Extracting domain models from textual requirements in the era of large language models, *MDEIntelligence (co-located with ACM/IEEE 26th International Conference on ...)*, 2023.
- [50] S. Ezzini, S. Abualhaija, C. Arora, M. Sabetzadeh, Automated handling of anaphoric ambiguity in requirements: a multi-solution study, in: *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 187–199.

- [51] A. Moharil, A. Sharma, Tabasco: A transformer based contextualization toolkit, *Science of Computer Programming* 230 (2023) 102994. doi:<https://doi.org/10.1016/j.scico.2023.102994>.
- [52] C. Arora, J. Grundy, M. Abdelrazek, Advancing requirements engineering through generative ai: Assessing the role of llms (2023). arXiv:2310.13976.
- [53] J. Zhang, Y. Chen, N. Niu, C. Liu, A preliminary evaluation of chatgpt in requirements information retrieval, arXiv preprint arXiv:2304.12562 (2023).
- [54] G. De Vito, F. Palomba, C. Gravino, S. Di Martino, F. Ferrucci, et al., Echo: An approach to enhance use case quality exploiting large language models, in: *2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2023, pp. 53–60.
- [55] S. Abualhaija, C. Arora, A. Sleimi, L. C. Briand, Automated question answering for improved understanding of compliance requirements: A multi-document study, in: *2022 IEEE 30th International Requirements Engineering Conference (RE)*, IEEE, 2022, pp. 39–50.
- [56] S. Ezzini, S. Abualhaija, C. Arora, M. Sabetzadeh, Ai-based question answering assistance for analyzing natural-language requirements, in: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, IEEE Computer Society, Los Alamitos, CA, USA, 2023, pp. 1277–1289. doi:10.1109/ICSE48619.2023.00113.
- [57] T. Hey, J. Keim, A. Koziolk, W. F. Tichy, Norbert: Transfer learning for requirements classification, in: *2020 IEEE 28th International Requirements Engineering Conference (RE)*, 2020, pp. 169–179. doi:10.1109/RE48521.2020.00028.
- [58] X. Luo, Y. Xue, Z. Xing, J. Sun, Prcbert: Prompt learning for requirement classification using bert-based pretrained language models, in: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, ASE '22*, Association for Computing Machinery, New York, NY, USA, 2023. doi:10.1145/3551349.3560417.
- [59] B. Chen, K. Chen, S. Hassani, Y. Yang, D. Amyot, L. Lessard, G. Mussbacher, M. Sabetzadeh, D. Varró, On the use of gpt-4 for creating goal

models: An exploratory study, in: MoDRE workshop at Requirement Engineering conference. To appear, 2023.

- [60] D. Hidellaarachchi, J. Grundy, R. Hoda, I. Mueller, The influence of human aspects on requirements engineering-related activities: Software practitioners' perspective, *ACM Transactions on Software Engineering and Methodology* 32 (5) (2023) 1–37.
- [61] Y. Li, S. Zhang, J. Sun, W. Zhang, Y. Du, Y. Wen, X. Wang, W. Pan, Tackling cooperative incompatibility for zero-shot human-AI coordination. *arXiv:2306.03034 [cs]*, doi:10.48550/arXiv.2306.03034. URL <http://arxiv.org/abs/2306.03034>
- [62] I. Issaoui, N. Bouassida, H. Ben-Abdallah, A new approach for interactive design pattern recommendation, *Lecture Notes on Software Engineering* 3 (3) (2015) 173.
- [63] N. Nahar, T. S. U. Mahmud, K. Sakib, An improved behavioral matching for anti-pattern based abstract factory recommendation, in: 2016 5th International Conference on Informatics, Electronics and Vision (ICIEV), IEEE, 2016, pp. 35–40.
- [64] J. E. Van Engelen, H. H. Hoos, A survey on semi-supervised learning, *Machine learning* 109 (2) (2020) 373–440.
- [65] C. Bou, N. Laosen, E. Nantajeewarawat, Design pattern ranking based on the design pattern intent ontology, in: *Intelligent Information and Database Systems: 10th Asian Conference, ACIIDS 2018, Dong Hoi City, Vietnam, March 19-21, 2018, Proceedings, Part I* 10, Springer, 2018, pp. 25–35.
- [66] A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, M. S. Aktar, T. Mikkonen, Towards human-bot collaborative software architecting with chatgpt, in: *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering, EASE '23, Association for Computing Machinery, New York, NY, USA, 2023*, p. 279–285. doi:10.1145/3593434.3593468. URL <https://doi.org/10.1145/3593434.3593468>

- [67] S. Herold, C. Knieke, M. Schindler, A. Rausch, Towards improving software architecture degradation mitigation by machine learning.
URL <https://urn.kb.se/resolve?urn=urn:nbn:se:kau:diva-81120>
- [68] J. J. Maranhão Junior, F. F. Correia, E. Guerra, Can chatgpt suggest patterns? an exploratory study about answers given by ai-assisted tools to design problems, in: AI-Assisted Agile Software Development Workshop at XP2023 conference, Springer, 2023.
- [69] T. Stojanovic, S. D. Lazarević, The application of ChatGPT for identification of microservices 3 (1) 99–105.
- [70] R. Feldt, S. Kang, J. Yoon, S. Yoo, Towards autonomous testing agents via conversational large language models. arXiv:2306.05152 [cs], doi:10.48550/arXiv.2306.05152.
URL <http://arxiv.org/abs/2306.05152>
- [71] E. Gamma, R. Helm, R. Johnson, J. Vlissides, D. Patterns, Elements of reusable object-oriented software, Design Patterns (1995).
- [72] L. Viviani, E. Guerra, J. Melegati, X. Wang, An empirical study about the instability and uncertainty of non-functional requirements, in: International Conference on Agile Software Development, Springer Nature Switzerland Cham, 2023, pp. 77–93.
- [73] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. El-nashar, J. Spencer-Smith, D. C. Schmidt, A prompt pattern catalog to enhance prompt engineering with ChatGPT. arXiv:2302.11382 [cs], doi:10.48550/arXiv.2302.11382.
URL <http://arxiv.org/abs/2302.11382>
- [74] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, L. Sifre, Training compute-optimal large language models (2022). arXiv:2203.15556.
- [75] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez,

- A. Joulin, E. Grave, G. Lample, Llama: Open and efficient foundation language models (2023). arXiv:2302.13971.
- [76] Andersen v. stability ai ltd. on courtlistener, accessed on 2023-10-05.
URL <https://www.courtlistener.com/docket/66732129/parties/andersen-v-stabili>
 - [77] Doe v. github inc. on courtlistener, accessed on 2023-10-05.
 - [78] A. Moradi Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, Z. M. J. Jiang, GitHub copilot AI pair programmer: Asset or liability? 203 111734. doi:10.1016/j.jss.2023.111734.
 - [79] S. Imai, Is GitHub copilot a substitute for human pair-programming? an empirical study, in: 2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pp. 319–321, ISSN: 2574-1926. doi:10.1145/3510454.3522684.
 - [80] M. L. Siddiq, A. Samee, S. R. Azgor, M. A. Haider, S. I. Sawraz, J. C. S. Santos, Zero-shot prompting for code complexity prediction using GitHub copilot, in: 2023 IEEE/ACM 2nd International Workshop on Natural Language-Based Software Engineering (NLBSE), pp. 56–59. doi:10.1109/NLBSE59153.2023.00018.
 - [81] S. Jiang, Y. Wang, Y. Wang, SelfEvolve: A code evolution framework via large language models. arXiv:2306.02907 [cs], doi:10.48550/arXiv.2306.02907.
URL <http://arxiv.org/abs/2306.02907>
 - [82] Y. Dong, X. Jiang, Z. Jin, G. Li, Self-collaboration code generation via chatgpt (2023). arXiv:2304.07590.
 - [83] A. Borji, A categorical archive of ChatGPT failures. arXiv:2302.03494 [cs], doi:10.48550/arXiv.2302.03494.
URL <http://arxiv.org/abs/2302.03494>
 - [84] 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: 27th International Conference on Intelligent User Interfaces, IUI '22, Association for Computing Machinery, pp. 212–228. doi:10.1145/3490099.3511119.
URL <https://dl.acm.org/doi/10.1145/3490099.3511119>

- [85] H. Li, Y. Hao, Y. Zhai, Z. Qian, The hitchhiker’s guide to program analysis: A journey with large language models (2023). arXiv:2308.00245.
- [86] B. Yetiştiren, I. Özsoy, M. Ayerdem, E. Tüzün, Evaluating the code quality of AI-assisted code generation tools: An empirical study on GitHub copilot, amazon CodeWhisperer, and ChatGPT. arXiv:2304.10778 [cs], doi:10.48550/arXiv.2304.10778.
URL <http://arxiv.org/abs/2304.10778>
- [87] P. Salza, C. Schwizer, J. Gu, H. C. Gall, On the effectiveness of transfer learning for code search 49 (4) 1804–1822. arXiv:2108.05890 [cs], doi:10.1109/TSE.2022.3192755.
URL <http://arxiv.org/abs/2108.05890>
- [88] F. Chen, F. Fard, D. Lo, T. Bryksin, On the transferability of pre-trained language models for low-resource programming languages. arXiv:2204.09653 [cs], doi:10.48550/arXiv.2204.09653.
URL <http://arxiv.org/abs/2204.09653>
- [89] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou, CodeBERT: A pre-trained model for programming and natural languages. arXiv:2002.08155 [cs], doi:10.48550/arXiv.2002.08155.
URL <http://arxiv.org/abs/2002.08155>
- [90] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. K. Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, M. Zhou, GraphCodeBERT: Pre-training code representations with data flow. arXiv:2009.08366 [cs], doi:10.48550/arXiv.2009.08366.
URL <http://arxiv.org/abs/2009.08366>
- [91] W. Gu, Z. Li, C. Gao, C. Wang, H. Zhang, Z. Xu, M. R. Lyu, CRaDLe: Deep code retrieval based on semantic dependency learning 141 385–394. arXiv:2012.01028 [cs], doi:10.1016/j.neunet.2021.04.019.
URL <http://arxiv.org/abs/2012.01028>
- [92] D. Li, Y. Shen, R. Jin, Y. Mao, K. Wang, W. Chen, Generation-augmented query expansion for code retrieval. arXiv:2212.10692 [cs],

doi:10.48550/arXiv.2212.10692.

URL <http://arxiv.org/abs/2212.10692>

- [93] M. Wei, N. S. Harzevili, Y. Huang, J. Wang, S. Wang, CLEAR: Contrastive learning for API recommendation, in: 2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE), pp. 376–387. doi:10.1145/3510003.3510159.
URL <https://ieeexplore.ieee.org/document/9793955>
- [94] K. Zhang, H. Zhang, G. Li, J. Li, Z. Li, Z. Jin, ToolCoder: Teach code generation models to use API search tools. arXiv:2305.04032 [cs], doi:10.48550/arXiv.2305.04032.
URL <http://arxiv.org/abs/2305.04032>
- [95] S. G. Patil, T. Zhang, X. Wang, J. E. Gonzalez, Gorilla: Large language model connected with massive APIs. arXiv:2305.15334 [cs], doi:10.48550/arXiv.2305.15334.
URL <http://arxiv.org/abs/2305.15334>
- [96] A. Mastropaolo, S. Scalabrino, N. Cooper, D. Nader Palacio, D. Poshyanyk, R. Oliveto, G. Bavota, Studying the usage of text-to-text transfer transformer to support code-related tasks, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021, pp. 336–347. doi:10.1109/ICSE43902.2021.00041.
- [97] M. Geng, S. Wang, D. Dong, H. Wang, G. Li, Z. Jin, X. Mao, X. Liao, Large language models are few-shot summarizers: Multi-intent comment generation via in-context learning. arXiv:2304.11384 [cs], doi:10.48550/arXiv.2304.11384.
URL <http://arxiv.org/abs/2304.11384>
- [98] P. Vaithilingam, T. Zhang, E. L. Glassman, Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models, in: Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems, CHI EA '22, Association for Computing Machinery, pp. 1–7. doi:10.1145/3491101.3519665.
URL <https://dl.acm.org/doi/10.1145/3491101.3519665>
- [99] S. Barke, M. B. James, N. Polikarpova, Grounded copilot: How programmers interact with code-generating models 7 78:85–78:111.

doi:10.1145/3586030.

URL <https://dl.acm.org/doi/10.1145/3586030>

- [100] A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, E. Aftandilian, Productivity assessment of neural code completion, in: Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, MAPS 2022, Association for Computing Machinery, pp. 21–29. doi:10.1145/3520312.3534864. URL <https://dl.acm.org/doi/10.1145/3520312.3534864>
- [101] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, R. Karri, Asleep at the keyboard? assessing the security of GitHub copilot’s code contributions, in: 2022 IEEE Symposium on Security and Privacy (SP), pp. 754–768, ISSN: 2375-1207. doi:10.1109/SP46214.2022.9833571.
- [102] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, I. Gazit, Taking flight with copilot: Early insights and opportunities of AI-powered pair-programming tools 20 (6) Pages 10:35–Pages 10:57. doi:10.1145/3582083. URL <https://dl.acm.org/doi/10.1145/3582083>
- [103] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, Lora: Low-rank adaptation of large language models (2021). arXiv:2106.09685.
- [104] X. L. Li, P. Liang, Prefix-tuning: Optimizing continuous prompts for generation, in: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Association for Computational Linguistics, Online, 2021, pp. 4582–4597. doi:10.18653/v1/2021.acl-long.353. URL <https://aclanthology.org/2021.acl-long.353>
- [105] X. Liu, K. Ji, Y. Fu, W. L. Tam, Z. Du, Z. Yang, J. Tang, P-tuning v2: Prompt tuning can be comparable to fine-tuning universally across scales and tasks (2022). arXiv:2110.07602.
- [106] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, J. Tang, Gpt understands, too (2021). arXiv:2103.10385.

- [107] B. Lester, R. Al-Rfou, N. Constant, The power of scale for parameter-efficient prompt tuning (2021). arXiv:2104.08691.
- [108] Z. Wang, R. Panda, L. Karlinsky, R. Feris, H. Sun, Y. Kim, Multitask prompt tuning enables parameter-efficient transfer learning (2023). arXiv:2303.02861.
- [109] H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, C. Raffel, Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning (2022). arXiv:2205.05638.
- [110] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shliazhko, N. Gontier, N. Meade, A. Zebaze, M.-H. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. Murthy, J. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, N. Fahmy, U. Bhattacharyya, W. Yu, S. Singh, S. Luccioni, P. Villegas, M. Kunakov, F. Zhdanov, M. Romero, T. Lee, N. Timor, J. Ding, C. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, J. Robinson, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. von Werra, H. de Vries, Starcoder: may the source be with you! (2023). arXiv:2305.06161.
- [111] S. Planning, The economic impacts of inadequate infrastructure for software testing, National Institute of Standards and Technology 1 (2002).
- [112] TestRail, The 2023 software testing and quality report (2023).
- [113] H. Liu, L. Liu, C. Yue, Y. Wang, B. Deng, Autotestgpt: A system for the automated generation of software test cases based on chatgpt, Available at SSRN 4584792.
- [114] Z. Yuan, Y. Lou, M. Liu, S. Ding, K. Wang, Y. Chen, X. Peng, No more manual tests? evaluating and improving chatgpt for unit test generation, arXiv preprint arXiv:2305.04207 (2023).

- [115] Z. Kotti, R. Galanopoulou, D. Spinellis, Machine learning for software engineering: A tertiary study, *ACM Computing Surveys* 55 (12) (2023) 1–39.
- [116] S. Jalil, S. Rafi, T. D. LaToza, K. Moran, W. Lam, Chatgpt and software testing education: Promises & perils, in: *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE, 2023, pp. 4130–4137.
- [117] W. Ma, S. Liu, W. Wang, Q. Hu, Y. Liu, C. Zhang, L. Nie, Y. Liu, The scope of chatgpt in software engineering: A thorough investigation, *arXiv preprint arXiv:2305.12138* (2023).
- [118] J. White, S. Hays, Q. Fu, J. Spencer-Smith, D. C. Schmidt, Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design, *arXiv preprint arXiv:2303.07839* (2023).
- [119] M. A. Akbar, A. A. Khan, P. Liang, Ethical aspects of chatgpt in software engineering research, *arXiv preprint arXiv:2306.07557* (2023).
- [120] V. Garousi, M. V. Mäntylä, A systematic literature review of literature reviews in software testing, *Information and Software Technology* 80 (2016) 195–216.
- [121] J. Wang, Y. Huang, C. Chen, Z. Liu, S. Wang, Q. Wang, Software testing with large language model: Survey, landscape, and vision, *arXiv preprint arXiv:2307.07221* (2023).
- [122] Z. Zheng, K. Ning, J. Chen, Y. Wang, W. Chen, L. Guo, W. Wang, Towards an understanding of large language models in software engineering tasks (2023). *arXiv:2308.11396*.
- [123] R. Feldt, S. Kang, J. Yoon, S. Yoo, Towards autonomous testing agents via conversational large language models, *arXiv preprint arXiv:2306.05152* (2023).
- [124] A. Przybyłek, An empirical study on the impact of aspectj on software evolvability, *Empirical Software Engineering* 23 (4) (2018) 2018–2050.

- [125] S. Jarzabek, Effective software maintenance and evolution: A reuse-based approach, CRC Press, 2007.
- [126] N. Anquetil, K. M. de Oliveira, K. D. de Sousa, M. G. Batista Dias, Software maintenance seen as a knowledge management issue, *Information and Software Technology* 49 (5) (2007) 515–529. doi:<https://doi.org/10.1016/j.infsof.2006.07.007>.
- [127] D. Noever, K. Williams, Chatbots as fluent polyglots: Revisiting breakthrough code snippets. arXiv:2301.03373 [cs], doi:10.48550/arXiv.2301.03373.
- [128] T. Ahmed, P. Devanbu, Few-shot training llms for project-specific code-summarization, in: *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, ASE '22*, Association for Computing Machinery, New York, NY, USA, 2023. doi:10.1145/3551349.3559555. URL <https://doi.org/10.1145/3551349.3559555>
- [129] A. Madaan, A. Shypula, U. Alon, M. Hashemi, P. Ranganathan, Y. Yang, G. Neubig, A. Yazdanbakhsh, Learning performance-improving code edits (2023). arXiv:2302.07867.
- [130] Q. Zhang, C. Fang, W. Sun, Y. Liu, T. He, X. Hao, Z. Chen, Boosting automated patch correctness prediction via pre-trained language model. arXiv:2301.12453 [cs], doi:10.48550/arXiv.2301.12453. URL <http://arxiv.org/abs/2301.12453>
- [131] C. S. Xia, Y. Wei, L. Zhang, Practical program repair in the era of large pre-trained language models (2022). arXiv:2210.14179.
- [132] H. Pearce, B. Tan, B. Ahmad, R. Karri, B. Dolan-Gavitt, Examining zero-shot vulnerability repair with large language models, in: *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 2339–2356. doi:10.1109/SP46215.2023.10179324.
- [133] O. Asare, M. Nagappan, N. Asokan, Is github’s copilot as bad as humans at introducing vulnerabilities in code? (2023). arXiv:2204.04741.

- [134] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, H. Wang, Large language models for software engineering: A systematic literature review (2023). arXiv:2308.10620.
- [135] I. Sommerville, Software Engineering, 10th Edition, Pearson.
- [136] A. Nguyen-Duc, S. M. A. Shah, P. Ambrahamsson, Towards an early stage software startups evolution model, in: 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 120–127. doi:10.1109/SEAA.2016.21.
- [137] B. Fitzgerald, K.-J. Stol, Continuous software engineering: A roadmap and agenda, Journal of Systems and Software 123 (2017) 176–189.
- [138] Y. Feng, S. Vanam, M. Cherukupally, W. Zheng, M. Qiu, H. Chen, Investigating code generation performance of ChatGPT with crowdsourcing social data, in: 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 876–885, ISSN: 0730-3157. doi:10.1109/COMPSAC57700.2023.00117.
- [139] J. D. Weisz, M. Muller, S. Houde, J. Richards, S. I. Ross, F. Martinez, M. Agarwal, K. Talamadupula, Perfection not required? human-AI partnerships in code translation, in: 26th International Conference on Intelligent User Interfaces, IUI '21, Association for Computing Machinery, pp. 402–412. doi:10.1145/3397481.3450656. URL <https://doi.org/10.1145/3397481.3450656>
- [140] S. I. Ross, F. Martinez, S. Houde, M. Muller, J. D. Weisz, The programmer’s assistant: Conversational interaction with a large language model for software development, in: Proceedings of the 28th International Conference on Intelligent User Interfaces, IUI '23, Association for Computing Machinery, pp. 491–514. doi:10.1145/3581641.3584037. URL <https://dl.acm.org/doi/10.1145/3581641.3584037>
- [141] N. Petrovic, Machine learning-based run-time DevSecOps: ChatGPT against traditional approach, in: 2023 10th International Conference on Electrical, Electronic and Computing Engineering (IcETRAN), pp. 1–5. doi:10.1109/IcETRAN59631.2023.10192161.

- [142] P. L. Lanzi, D. Loiacono, ChatGPT and other large language models as evolutionary engines for online interactive collaborative game design, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '23, Association for Computing Machinery, pp. 1383–1390. doi:10.1145/3583131.3590351.
- [143] Y. Ye, H. You, J. Du, Improved trust in human-robot collaboration with ChatGPT 11 55748–55754, conference Name: IEEE Access. doi:10.1109/ACCESS.2023.3282111.
- [144] N. Nascimento, P. Alencar, D. Cowan, Comparing software developers with ChatGPT: An empirical investigation. arXiv:2305.11837 [cs], doi:10.48550/arXiv.2305.11837.
URL <http://arxiv.org/abs/2305.11837>
- [145] H. Eisner, Essentials of project and systems engineering management, John Wiley & Sons, 2008.
- [146] P. Tambe, P. Cappelli, V. Yakubovich, Artificial intelligence in human resources management: Challenges and a path forward, California Management Review 61 (4) (2019) 15–42.
- [147] A. P. Ammeter, J. M. Dukerich, Leadership, team building, and team member characteristics in high performance project teams, Engineering management journal 14 (4) (2002) 3–10.
- [148] H. Khamooshi, D. F. Cioffi, Program risk contingency budget planning, IEEE Transactions on Engineering Management 56 (1) (2009) 171–179.
- [149] B. S. Blanchard, System engineering management, John Wiley & Sons, 2004.
- [150] N. J. Smith, Engineering project management, Blackwell Science Ames, IA, 2002.
- [151] L. Wang, Z. Liu, A. Liu, F. Tao, Artificial intelligence in product life-cycle management, The International Journal of Advanced Manufacturing Technology 114 (2021) 771–796.
- [152] V. Holzmann, D. Zitter, S. Peshkess, The expectations of project managers from artificial intelligence: A delphi study, Project Management Journal 53 (5) (2022) 438–455.

- [153] V. Prifti, Optimizing project management using artificial intelligence, *European Journal of Formal Sciences and Engineering* 5 (1) (2022) 29–37.
- [154] H. K. Dam, T. Tran, J. Grundy, A. Ghose, Y. Kamei, Towards effective ai-powered agile project management, in: 2019 IEEE/ACM 41st international conference on software engineering: new ideas and emerging results (ICSE-NIER), IEEE, 2019, pp. 41–44.
- [155] H. H. Elmousalami, Comparison of artificial intelligence techniques for project conceptual cost prediction: a case study and comparative analysis, *IEEE Transactions on Engineering Management* 68 (1) (2020) 183–196.
- [156] M. Fu, C. Tantithamthavorn, GPT2sp: A transformer-based agile story point estimation approach 49 (2) 611–625. doi:10.1109/TSE.2022.3158252.
URL <https://ieeexplore.ieee.org/document/9732669>
- [157] M. Alhamed, T. Storer, Evaluation of context-aware language models and experts for effort estimation of software maintenance issues, in: 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 129–138, ISSN: 2576-3148. doi:10.1109/ICSME55016.2022.00020.
URL <https://ieeexplore.ieee.org/document/9978209>
- [158] M.-N. Chu, Assessing the benefits of ChatGPT for business: An empirical study on organizational performance 11 76427–76436, conference Name: IEEE Access. doi:10.1109/ACCESS.2023.3297447.
- [159] C. Ebert, P. Louridas, Generative AI for software practitioners 40 (4) 30–38, conference Name: IEEE Software. doi:10.1109/MS.2023.3265877.
- [160] Y. Pan, L. Zhang, Roles of artificial intelligence in construction engineering and management: A critical review and future trends, *Automation in Construction* 122 (2021) 103517.
- [161] S. O. Abioye, L. O. Oyedele, L. Akanbi, A. Ajayi, J. M. D. Delgado, M. Bilal, O. O. Akinade, A. Ahmed, Artificial intelligence in the con-

struction industry: A review of present status, opportunities and future challenges, *Journal of Building Engineering* 44 (2021) 103299.

- [162] N. A. Parikh, Empowering business transformation: The positive impact and ethical considerations of generative ai in software product management—a systematic literature review, *arXiv preprint arXiv:2306.04605* (2023).
- [163] A. Brem, F. Giones, M. Werle, The ai digital revolution in innovation: A conceptual framework of artificial intelligence technologies for the management of innovation, *IEEE Transactions on Engineering Management* (2021).
- [164] L. Song, L. L. Minku, Artificial intelligence in software project management, in: *Optimising the Software Development Process with Artificial Intelligence*, Springer, 2023, pp. 19–65.
- [165] A. El-Deeb, The human side of the tech industry: Key drivers behind the tech talent dilemma, *SIGSOFT Softw. Eng. Notes* 47 (1) (2022) 10–11. doi:10.1145/3502771.3502775.
URL <https://doi.org/10.1145/3502771.3502775>
- [166] E. Sarikaya, S. Bagriyanik, M. Gökalp, Teaching agile software development using agile methods: A case study, in: *2020 Turkish National Software Engineering Symposium (UYMS)*, 2020, pp. 1–6. doi:10.1109/UYMS50627.2020.9247027.
- [167] S. Tennakoon, R. Islam, D. Wijerathna, R. Deen, P. Sumathipala, L. Abeywardhana, An interactive application for university students to reduce the industry-academia skill gap in the software engineering field, in: *2023 4th International Conference for Emerging Technology (INCET)*, 2023, pp. 1–6. doi:10.1109/INCET57972.2023.10170591.
- [168] S. Hyrynsalmi, E. Sutinen, The role of women software communities in attracting more women to the software industry, in: *2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, 2019, pp. 1–7. doi:10.1109/ICE.2019.8792673.
- [169] L. Jiagui, W. Wanli, J. F. I. Lam, L. Ke, SWOT analysis and public policy of macao’s digital trade in services 9 (1) 2176363.

doi:10.1080/23311886.2023.2176363.

URL <https://doi.org/10.1080/23311886.2023.2176363>

- [170] S. Verdugo-Castro, A. García-Holgado, M. C. Sánchez-Gómez, The gender gap in higher stem studies: a systematic literature review, *Helvion* (2022).
- [171] R. S. Dubey, J. Paul, V. Tewari, The soft skills gap: a bottleneck in the talent supply in emerging economies, *The International Journal of Human Resource Management* 33 (13) (2022) 2630–2661. arXiv:<https://doi.org/10.1080/09585192.2020.1871399>, doi:10.1080/09585192.2020.1871399. URL <https://doi.org/10.1080/09585192.2020.1871399>
- [172] J. Zhao, T. Wang, M. Yatskar, V. Ordonez, K.-W. Chang, Men also like shopping: Reducing gender bias amplification using corpus-level constraints (2017). arXiv:1707.09457.
- [173] OpenAI, Gpt-4 technical report (2023). arXiv:2303.08774.
- [174] D. Adiwardana, M. Luong, D. R. So, J. Hall, N. Fiedel, R. Thoppilan, Z. Yang, A. Kulshreshtha, G. Nemade, Y. Lu, Q. V. Le, Towards a human-like open-domain chatbot, *CoRR* abs/2001.09977 (2020). arXiv:2001.09977. URL <https://arxiv.org/abs/2001.09977>
- [175] European Commission, Directorate-General for Communications Networks, Content and Technology, EUR-Lex - 52021PC0206 - EN - EUR-Lex, CNECT (2021).
- [176] L. A. González, A. Neyem, I. Contreras-McKay, D. Molina, Improving learning experiences in software engineering capstone courses using artificial intelligence virtual assistants 30 (5) 1370–1389. doi:10.1002/cae.22526.
- [177] M. Verleger, J. Pembrige, A Pilot Study Integrating an AI-driven Chatbot in an Introductory Programming Course, *IEEE*, 2018, pp. 1–4. doi:10.1109/FIE.2018.8659282. URL <https://ieeexplore.ieee.org/document/8659282/>

- [178] M. Binkis, R. Kubiliūnas, R. Sturienė, T. Dulinskienė, T. Blažauskas, V. Jakštienė, Rule-based chatbot integration into software engineering course, in: A. Lopata, D. Gudonienė, R. Butkienė (Eds.), *Information and Software Technologies, Communications in Computer and Information Science*, Springer International Publishing, pp. 367–377.
- [179] M. Ismail, A. Ade-Ibijola, Lecturer’s apprentice: A chatbot for assisting novice programmers, *IEEE*, 2019, pp. 1–8. doi:10.1109/IMITEC45504.2019.9015857.
URL <https://ieeexplore.ieee.org/document/9015857/>
- [180] G. Carreira, L. Silva, A. J. Mendes, H. G. Oliveira, Pyo, a chatbot assistant for introductory programming students, *IEEE*, 2022, pp. 1–6. doi:10.1109/SIIE56031.2022.9982349.
URL <https://ieeexplore.ieee.org/document/9982349/>
- [181] J. C. Paiva, J. P. Leal, A. Figueira, Automated assessment in computer science education: A state-of-the-art review 22 (3) 34:1–34:40.
- [182] M. Cukusic, Z. Garaca, M. Jadric, Online self-assessment and students’ success in higher education institutions 72 100–109. doi:10.1016/j.compedu.2013.10.018.
- [183] J. Qadir, Engineering education in the era of ChatGPT: Promise and pitfalls of generative AI for education, in: 2023 IEEE Global Engineering Education Conference (EDUCON), pp. 1–9. doi:10.1109/EDUCON54358.2023.10125121.
URL <https://ieeexplore.ieee.org/document/10125121>
- [184] M. Daun, J. Brings, How ChatGPT will change software engineering education, in: *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1*, ITiCSE 2023, Association for Computing Machinery, pp. 110–116. doi:10.1145/3587102.3588815.
URL <https://doi.org/10.1145/3587102.3588815>
- [185] C. Bull, A. Kharrufa, Generative AI assistants in software development education: A vision for integrating generative AI into educational practice, not instinctively defending against it. 1–9Conference Name: IEEE Software. doi:10.1109/MS.2023.3300574.

- [186] J. Berrezueta-Guzman, S. Krusche, Recommendations to create programming exercises to overcome ChatGPT, in: 2023 IEEE 35th International Conference on Software Engineering Education and Training (CSEET), pp. 147–151, ISSN: 2377-570X. doi:10.1109/CSEET58097.2023.00031.
- [187] B. Banić, M. Konecki, M. Konecki, Pair programming education aided by ChatGPT, in: 2023 46th MIPRO ICT and Electronics Convention (MIPRO), pp. 911–915, ISSN: 2623-8764. doi:10.23919/MIPRO57284.2023.10159727.
- [188] S. Jalil, S. Rafi, T. D. LaToza, K. Moran, W. Lam, ChatGPT and software testing education: Promises & perils, in: 2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 4130–4137. doi:10.1109/ICSTW58534.2023.00078.
- [189] A. Ashraf, A. Imam, ChatGPT’s use case for software engineers, in: 8th International Conference on Computing in Engineering and Technology (ICCET 2023), Vol. 2023, pp. 487–492. doi:10.1049/icp.2023.1537.
- [190] J. Savelka, A. Agarwal, C. Bogart, M. Sakr, Large language models (GPT) struggle to answer multiple-choice questions about code. arXiv:2303.08033 [cs], doi:10.48550/arXiv.2303.08033. URL <http://arxiv.org/abs/2303.08033>
- [191] R. Yilmaz, F. G. Karaoglan Yilmaz, The effect of generative artificial intelligence (AI)-based tool use on students’ computational thinking skills, programming self-efficacy and motivation 4 100147. doi:10.1016/j.caeai.2023.100147.
- [192] C. A. Philbin, Exploring the potential of artificial intelligence program generators in computer programming education for students 14 (3) 30–38. doi:10.1145/3610406. URL <https://dl.acm.org/doi/10.1145/3610406>
- [193] M. Ewens, R. Nanda, M. Rhodes-Kropf, Cost of experimentation and the evolution of venture capital, *Journal of Financial Economics* 128 (3) (2018) 422–442. doi:10.1016/j.jfineco.2018.03.001.
- [194] S. Blank, *The Four Steps to the Epiphany: Successful Strategies for Products that Win*, Cafepress.com, 2007.

- [195] E. Ries, *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses*, The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Business, 2011.
- [196] V. Vakkuri, K.-K. Kemell, J. Kultanen, P. Abrahamsson, The Current State of Industrial Practice in Artificial Intelligence Ethics, *IEEE Software* 37 (4) (2020) 50–57. doi:10.1109/MS.2020.2985621.
- [197] S. Morandini, t. l. w. o. i. a. n. t. Link to external site, F. Fraboni, t. l. w. o. i. a. n. t. Link to external site, M. De Angelis, t. l. w. o. i. a. n. t. Link to external site, G. Puzzo, t. l. w. o. i. a. n. t. Link to external site, D. Giusino, t. l. w. o. i. a. n. t. Link to external site, L. Pietrantoni, t. l. w. o. i. a. n. t. Link to external site, The impact of artificial intelligence on workers' skills: Upskilling and reskilling in organisations 26 39–68. doi:10.28945/5078.
- [198] D. K. Kanbach, L. Heiduk, G. Blueher, M. Schreiter, A. Lahmann, The GenAI is out of the bottle: generative artificial intelligence from a business model innovation perspective, *Review of Managerial Science* (0123456789) (sep 2023).
- [199] B. Meskó, E. J. Topol, The imperative for regulatory oversight of large language models (or generative AI) in healthcare, *npj Digital Medicine* 6 (1) (2023) 120. doi:10.1038/s41746-023-00873-0.
- [200] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, T. Zimmermann, Software engineering for machine learning: A case study, in: *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, IEEE, 2019, pp. 291–300.
- [201] W. Ma, S. Liu, W. Wang, Q. Hu, Y. Liu, C. Zhang, L. Nie, Y. Liu, The scope of ChatGPT in software engineering: A thorough investigation. arXiv:2305.12138 [cs], doi:10.48550/arXiv.2305.12138. URL <http://arxiv.org/abs/2305.12138>
- [202] R. Azamfirei, S. R. Kudchadkar, J. Fackler, Large language models and the perils of their hallucinations 27 (1) 120. doi:10.1186/s13054-023-04393-x. URL <https://doi.org/10.1186/s13054-023-04393-x>

- [203] P. Manakul, A. Liusie, M. J. F. Gales, SelfCheckGPT: Zero-resource black-box hallucination detection for generative large language models. arXiv:2303.08896 [cs], doi:10.48550/arXiv.2303.08896.
URL <http://arxiv.org/abs/2303.08896>
- [204] C. Barrett, B. Boyd, E. Burzstein, N. Carlini, B. Chen, J. Choi, A. R. Chowdhury, M. Christodorescu, A. Datta, S. Feizi, K. Fisher, T. Hashimoto, D. Hendrycks, S. Jha, D. Kang, F. Kerschbaum, E. Mitchell, J. Mitchell, Z. Ramzan, K. Shams, D. Song, A. Taly, D. Yang, Identifying and mitigating the security risks of generative AI. arXiv:2308.14840 [cs], doi:10.48550/arXiv.2308.14840.
URL <http://arxiv.org/abs/2308.14840>
- [205] L. Chen, M. Zaharia, J. Zou, How is chatgpt’s behavior changing over time? (2023). arXiv:2307.09009.
- [206] A. Hamid, H. R. Samidi, T. Finin, P. Pappachan, R. Yus, GenAIPABench: A benchmark for generative AI-based privacy assistants. arXiv:2309.05138 [cs], doi:10.48550/arXiv.2309.05138.
URL <http://arxiv.org/abs/2309.05138>
- [207] E. Kasneci, K. Sessler, S. Kuchemann, M. Bannert, D. Dementieva, F. Fischer, U. Gasser, G. Groh, S. Gunnemann, E. Hullermeier, S. Krusche, G. Kutyniok, T. Michaeli, C. Nerdel, J. Pfeffer, O. Poquet, M. Sailer, A. Schmidt, T. Seidel, M. Stadler, J. Weller, J. Kuhn, G. Kasneci, ChatGPT for good? on opportunities and challenges of large language models for education 103 102274. doi:10.1016/j.lindif.2023.102274.
- [208] M. T. Ribeiro, S. Singh, C. Guestrin, ”why should i trust you?”: Explaining the predictions of any classifier (2016). arXiv:1602.04938.
- [209] D. Levin, R. Cross, L. Abrams, Why should i trust you? predictors of interpersonal trust in a knowledge transfer context (09 2002).
- [210] B. Cabrero-Daniel, A. Sanagustín Cabrero, Perceived trustworthiness of natural language generators, in: Proceedings of the First International Symposium on Trustworthy Autonomous Systems, TAS ’23, Association for Computing Machinery, New York, NY, USA, 2023.

doi:10.1145/3597512.3599715.

URL <https://doi.org/10.1145/3597512.3599715>

- [211] N. Amsel, Z. Ibrahim, A. Malik, B. Tomlinson, Toward sustainable software engineering (NIER track), in: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, Association for Computing Machinery, pp. 976–979. doi:10.1145/1985793.1985964. URL <https://dl.acm.org/doi/10.1145/1985793.1985964>
- [212] B. Penzenstadler, V. Bauer, C. Calero, X. Franch, Sustainability in software engineering: a systematic literature review 32–41doi:10.1049/ic.2012.0004.
- [213] S. Naumann, E. Kern, M. Dick, T. Johann, Sustainable software engineering: Process and quality models, life cycle, and social aspects, in: L. M. Hilty, B. Aebischer (Eds.), ICT Innovations for Sustainability, Advances in Intelligent Systems and Computing, Springer International Publishing, pp. 191–205.
- [214] A. A. Chien, GenAI: Giga\$\$\$, TeraWatt-hours, and GigaTons of CO2 66 (8) 5. doi:10.1145/3606254. URL <https://dl.acm.org/doi/10.1145/3606254>
- [215] A. A. Chien, L. Lin, H. Nguyen, V. Rao, T. Sharma, R. Wijayawardana, Reducing the carbon impact of generative AI inference (today and in 2035), in: Proceedings of the 2nd Workshop on Sustainable Computer Systems, HotCarbon '23, Association for Computing Machinery, pp. 1–7. doi:10.1145/3604930.3605705. URL <https://dl.acm.org/doi/10.1145/3604930.3605705>
- [216] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, J. Pineau, Towards the systematic reporting of the energy and carbon footprints of machine learning. arXiv:2002.05651 [cs], doi:10.48550/arXiv.2002.05651. URL <http://arxiv.org/abs/2002.05651>
- [217] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, D. Amodei, Scaling laws for neural language models. arXiv:2001.08361 [cs, stat], doi:10.48550/arXiv.2001.08361.

- [218] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, E. Rutherford, T. Hennigan, J. Menick, A. Cassirer, R. Powell, G. v. d. Driessche, L. A. Hendricks, M. Rauh, P.-S. Huang, A. Glaese, J. Welbl, S. Dathathri, S. Huang, J. Uesato, J. Mellor, I. Higgins, A. Creswell, N. McAleese, A. Wu, E. Elsen, S. Jayakumar, E. Buchatskaya, D. Budden, E. Sutherland, K. Simonyan, M. Paganini, L. Sifre, L. Martens, X. L. Li, A. Kuncoro, A. Nematzadeh, E. Gribovskaya, D. Donato, A. Lazaridou, A. Mensch, J.-B. Lespiau, M. Tsimpoukelli, N. Grigorev, D. Fritz, T. Sottiaux, M. Pajarskas, T. Pohlen, Z. Gong, D. Toyama, C. d. M. d’Autume, Y. Li, T. Terzi, V. Mikulik, I. Babuschkin, A. Clark, D. d. L. Casas, A. Guy, C. Jones, J. Bradbury, M. Johnson, B. Hechtman, L. Weidinger, I. Gabriel, W. Isaac, E. Lockhart, S. Osindero, L. Rimell, C. Dyer, O. Vinyals, K. Ayoub, J. Stanway, L. Bennett, D. Hasabis, K. Kavukcuoglu, G. Irving, Scaling language models: Methods, analysis & insights from training gopher. arXiv:2112.11446 [cs], doi:10.48550/arXiv.2112.11446.