Trustworthy and synergistic artificial intelligence for software engineering: Vision and roadmaps

David LO
Singapore Management University, davidlo@smu.edu.sg

Citation

Trustworthy and Synergistic Artificial Intelligence for Software Engineering: Vision and Roadmaps

David Lo

School of Computing and Information Systems,
Singapore Management University,
Singapore
Email: davidlo@smu.edu.sg

Abstract—For decades, much software engineering research has been dedicated to devising automated solutions aimed at enhancing developer productivity and elevating software quality. The past two decades have witnessed an unparalleled surge in the development of intelligent solutions tailored for software engineering tasks. This momentum established the Artificial Intelligence for Software Engineering (AI4SE) area, which has swiftly become one of the most active and popular areas within the software engineering field.

This Future of Software Engineering (FoSE) paper navigates through several focal points. It commences with a succinct introduction and history of AI4SE. Thereafter, it underscores the core challenges inherent to AI4SE, particularly highlighting the need to realize trustworthy and synergistic AI4SE. Progressing, the paper paints a vision for the potential leaps achievable if AI4SE's key challenges are surmounted, suggesting a transition toward Software Engineering 2.0. Two strategic roadmaps are then laid out: one centered on realizing trustworthy AI4SE, and the other on fostering synergistic AI4SE. While this paper may not serve as a conclusive guide, its intent is to catalyze further progress. The ultimate aspiration is to position AI4SE as a linchpin in redefining the horizons of software engineering, propelling us toward Software Engineering 2.0.

Index Terms—AI4SE, Trustworthy AI, Human-AI Collaboration, Software Engineering 2.0, Vision, Roadmaps

Fig. 1: Overview of the Paper with Its Six Sections

I. INTRODUCTION AND BRIEF HISTORY OF AI4SE

"*Study the past if you would define the future.*" – Confucius

Software engineering encompasses many tasks spanning the various phases of software development, from requirement gathering and design to coding, testing, and deployment. To boost developer productivity and ensure high-quality software, extensive research in software engineering has aimed to automate some of these manual tasks. While initial automation efforts centered around the development of program analysis methods, e.g., linters [1], model checkers [2], [3], fuzzers [4], etc., the past two decades have witnessed a rapid rise in the design and deployment of AI-powered solutions to assist software practitioners in their tasks.

AI-powered solutions have been employed to analyze a myriad of software artifacts, both products and by-products of software engineering activities. These artifacts encompass source code, execution traces, bug reports, and posts on question-and-answer sites, among others. A variety of AI methods underpin the development of these assistive solutions. In this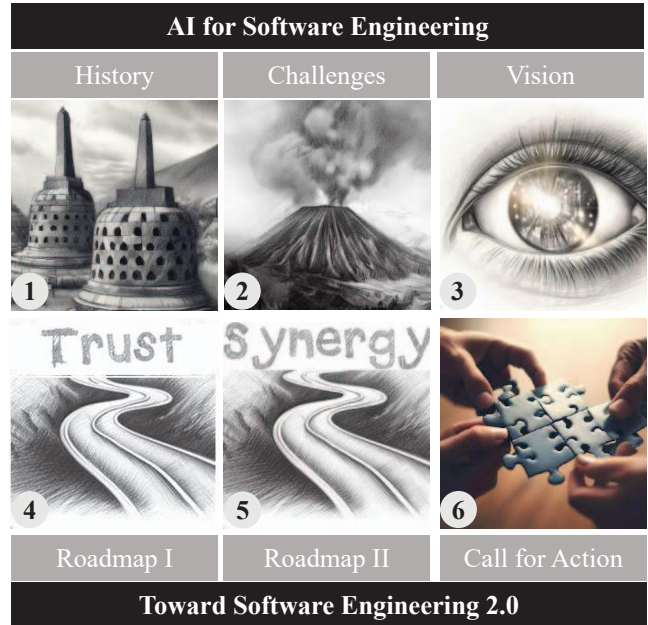 paper, the term "AI" is employed in a broad sense, encompassing a range of techniques from data mining and information retrieval to meta-heuristics, natural language processing, and machine learning. Today, the boundaries between these fields are becoming increasingly blurred, as many emerging techniques span across them.

As illustrated in Fig. 1, this paper first describes the history and key challenges of AI for Software Engineering (AI4SE) in Sections I and II, respectively. In then describes a vision for AI4SE in Section III. Next, it highlights two roadmaps toward trustworthy and synergistic AI4SE in Sections IV and V, respectively. The paper concludes by providing a summary and a call for action in Section VI.

The remainder of this section offers a concise overview of AI4SE's history after "AI winters" ended in the mid-2000s.[1] Even focusing from the mid-2000s to today, there are too many studies to cover. Thus, this section does not attempt to provide comprehensive coverage. For more coverage of the

---

[1] "Scientists and writers describe the dashed hopes of the period from the 1970s until the mid-2000s as a series of 'AI winters.'" [5]

history of AI4SE, please refer to other reviews and surveys. An example review of the emergence of AI4SE after the "AI winters" is by Xie et al. [6] on data mining for software engineering. The article described the applications of four AI techniques – frequent pattern mining, pattern matching, clustering, and classification – to analyze software engineering artifacts such as execution traces, source code, and bug reports, which can be represented in forms like sequences, graphs, and text. Notably, it described specialized AI techniques adept at inferring software's formal specifications (e.g., [7]), synthesizing bug signatures (e.g., [8]), and identifying duplicate bug reports (e.g., [9]). Beyond this review, there are others covering various AI4SE topics, e.g., [10], [11].[2]

Over the past two decades, significant efforts have been made to solidify AI4SE as a recognized research area within the software engineering field. A prominent one is the Mining Software Repositories (MSR) conference series, inaugurated as a workshop at the 26th ACM/IEEE International Conference on Software Engineering (ICSE 2004). This workshop "brought together researchers and practitioners in order to consider methods that use data stored in software repositories (such as source control systems, defect tracking systems, and archived project communications) to further understanding of software development practices." [13] Another prominent conference series is the Symposium on Search Based Software Engineering (SSBSE), which started in 2009 [14], and serves as a forum that focuses on solving software engineering tasks by formulating them as optimization or search problems.

A number of AI4SE workshops have also made significant contributions. One that ran for many years (2012 to 2018) was the International Workshop on Software Mining (SoftMine). SoftMine "facilitated researchers who are interested in mining various types of software-related data and in applying data mining techniques to support software engineering tasks." [15] By being hosted at both SE and AI conferences – including the IEEE/ACM International Conference on Automated Software Engineering (ASE) and the ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD) – SoftMine bridged the gap between the SE and AI communities. Additionally, numerous AI4SE tutorials and summer/winter schools have been organized over the years.

Today, these community efforts have borne fruit, evidenced by the prominence of AI4SE at leading software engineering conferences. It is noteworthy that at the 45th ACM/IEEE International Conference on Software Engineering (ICSE 2023), "Artificial Intelligence and Software Engineering" and "Software Analytics" were not only among the conference's primary seven areas but also among its most popular.

There have been multiple "waves" that positively influence the area of AI4SE:

Wave 1 (Big Software Engineering Data)

By the late 2000s, an increasing volume of open-source data became accessible to researchers, marking a significant shift from the time when most software artifacts remained confined within corporate boundaries. Notably, 2008 witnessed the launch of platforms like GitHub and Stack Overflow. The repositories on GitHub have expanded consistently over the years, and Stack Overflow's post count has shown a similar trajectory. These platforms supply vast amounts of data for AI4SE. As early as 2013, SE researchers had analyzed tens of thousands of version control and issue tracking systems from GitHub repositories [16]–[18]. Additionally, these platforms have presented unique challenges that demand AI4SE solutions. For example, the high number of posts in Stack Overflow requires the design of solutions that can empower the community to better browse [19], [20], locate [21], [22], maintain [23], [24], and comprehend [25], [26] answers to software engineering questions.

Wave 2 (Deep Learning for Software Engineering)

By the 2010s, deep learning has gained much traction across various domains, beyond computer vision. This evolution significantly influenced the AI4SE landscape. Initial forays into integrating deep learning with software engineering include constructing deep learning solutions for defect prediction [27] and code suggestion [28]. Subsequent research expanded upon various deep learning architectures, such as the Recurrent Neural Network and Transformer, aiming to automate a wider range of software engineering tasks. Moreover, considerable effort went into devising methods to learn effective distributed representations of diverse software artifacts, exemplified by works like [29], [30]. Comprehensive surveys on this topic were written by Yang et al. [31] and Watson et al. [32].

While there have been notable successes in marrying deep learning and software engineering, there are documented instances of limitations. Some studies, such as [33]–[35], demonstrate that, in certain situations and tasks, the efficacy of simpler techniques may be comparable to, or even surpass, deep learning models. Moreover, concerns about the generalizability of representations derived via deep learning are also highlighted by some studies such as [36].

Wave 3 (Large Language Model or Foundation Model for Software Engineering)

This wave continues from Wave 2. In 2018, Google introduced the Bidirectional Encoder Representations from Transformers (BERT), which underwent pre-training on the Toronto Book Corpus and English Wikipedia [37]. Often considered the first Large Language Model (LLM) or Foundation Model (FM)[3], BERT can significantly reduce the necessity for large amounts of high-quality labeled data for downstream tasks. By 2020, researchers had applied BERT to automate software engineering tasks [38], [39]. That same year, Microsoft unveiled CodeBERT, which is a variant of BERT pre-trained

---

[2]For a review on studies in the intersection of AI and software engineering research before the end of "AI winters", see, for instance, [12].

[3]The terms "Large Language Model" and "Foundation Model" are currently frequently used synonymously. However, a Foundation Model can encompass more than just a Large Language Model. For instance, it can support multiple modalities, including images. Since most software engineering studies today utilize Large Language Models as their Foundational Models, this paper predominantly uses the term "Large Language Model."

on a corpus containing a mixture of source code and natural language text [40]. After BERT, ever larger LLMs have been developed, with models like GPT-4 [41] currently leading the way. These LLMs have been leveraged to automate many software engineering tasks [42]. Interestingly, many years before the development of LLMs, Hindle et al. [43] have underscored the inherent naturalness of software and the prospective utility of language models in automating software engineering tasks.

The most recent wave has significantly advanced the adoption of AI4SE solutions among practitioners. Today, solutions like GitHub Copilot, Amazon CodeWhisperer, and OpenAI ChatGPT, built upon LLMs, are used by many professional and aspiring software practitioners. While, as of this paper's writing, their primary applications have been in coding, it is easy to envision their expansion into a broader spectrum of software engineering tasks in the near future, such as design, requirement elicitation, verification, and bug report management, among others.

II. KEY CHALLENGES OF AI4SE: TRUST AND SYNERGY

"Victory comes from finding opportunities in problems"
– Sun Tzu

While there has been a surge in AI4SE research and its widespread adoption, numerous challenges remain, offering ample opportunities for future research. Many of these challenges can be put into two broad categories: ensuring **trustworthy AI4SE** and promoting **synergistic AI4SE**. If AI4SE solutions are not trusted by practitioners, they will not be adopted. Trust is dynamic; AI4SE solutions need to maintain practitioners' confidence in them over time. Moreover, an effective AI4SE solution should not only be trustworthy but also synergize seamlessly with practitioners. If not, such AI4SE solutions risk becoming obstacles rather than facilitators. While trust and synergy are interconnected concepts, each has its unique characteristics.

A. Need for Trust

In 2015, a study involving 512 Microsoft practitioners was carried out to assess their perceptions on the relevance of software engineering research [44]. The goal was to identify potential gaps between academic research and its practical application. The results underscored concerns from practitioners, including those revolving on trust. For instance, one respondent stated "It seems that there could be potentially disastrous results if the automation does not [do things] correctly." A follow-up study in 2016, involving 386 practitioners from more than 30 countries across 5 continents, highlighted similar findings [45]. For example, one respondent stated, "I doubt any automated software can explain the reason for things . . .", highlighting a reason behind the lack of trust.

Fast forward to 2023, and although AI4SE research has undoubtedly advanced since 2015, challenges persist. A 2022 study revealed that many code snippets produced by GitHub Copilot contain security vulnerabilities [46]. Similarly, a 2023 article pointed out that code generated by ChatGPT often had

compilation and runtime errors, especially when applied to newer programming tasks that might not have been present in its training data [47]. Recent news articles further emphasize these concerns, such as "Friend or foe: Can computer coders trust ChatGPT?" [48] and "ChatGPT creates mostly insecure code, but won't tell you unless you ask" [49]. There are also concerns about significant variations in ChatGPT's efficacy over time [50], including in the code generation task. Such inconsistencies can undermine trust as practitioners seek stability in the efficacy of AI4SE solutions [51].

If these trust issues are not adequately addressed, the current enthusiasm surrounding AI4SE can possibly diminish, reminiscent of the declining interest in AI experienced during the "AI winters." [52]. Another side of the trust spectrum is over-reliance. Novices may mistakenly place excessive trust in these AI4SE solutions, expecting flawless results, while remaining oblivious to their inherent limitations, which can prove detrimental.

B. Need for Synergy

Synergy is typically defined as the collaboration of two or more entities to create an outcome that exceeds the sum of their individual contributions. In the context of AI4SE solutions, there are two primary usage scenarios: `1:1`, where a single software practitioner interacts with an AI4SE solution, and `N:M`, involving multiple software practitioners collaborating with multiple AI4SE solutions. Presently, the majority of research and existing AI4SE solutions focus on the `1:1` scenario, which is intuitively more straightforward than the `N:M` scenario. However, even within this simpler setting, there is no guarantee that software practitioners and AI4SE solutions will achieve seamless synergy.

Interlink between synergy and trust

Synergy and trust are intrinsically linked; it is challenging for two entities to work together seamlessly without trust. As an example, Parnin and Orso conducted a controlled experiment showing that while fault localization solutions[4] can achieve favorable results by certain metrics, they do not necessarily expedite human debugging processes [55]. Another research, although not centered on AI4SE, underscores that many professionals avoid static analysis solutions due to their frequent false positives, among other reasons [56], [57]. These studies underscore the "boy who cried wolf" phenomenon: when AI4SE solutions (or any automated solution for that matter) repeatedly produce unreliable results, software practitioners become skeptical and may disengage, preventing any synergy.

Once trust is firmly established, AI4SE solutions and practitioners can potentially harmonize in ways that yield tangible benefits. For instance, for fault localization, if an AI4SE solution consistently presents accurate outcomes within the

---

[4]A fault localization solution produces a ranked list of program locations that are likely to be faulty [53], [54]. They typically take as input a collection of program spectra describing program locations that are executed by failing and successful test cases.

top-5 or top-10 results, Xia et al. found that practitioners can gain a significant performance boost [58]. But the bar for adoption can be steep: Kochhar et al. noted that for 90% of practitioners to adopt fault localization solutions, these solutions must deliver accurate results within the top-5 positions at least 90% of the time [45]. Achieving such a goal is undeniably challenging.

This narrative emphasizes that synergy and trust are closely intertwined, with trust being influenced by the efficacy of AI4SE solutions, specifically their capability and likelihood to yield accurate results. Although slight enhancements in efficacy may not immediately foster trust, there exists a critical threshold that, once exceeded, can serve as a tipping point for both trust and synergy. Therefore, as a community, it is essential to continually push for greater efficacy, even if immediate gains in trust and synergy are not readily visible.

Synergy beyond trust

While synergy undoubtedly involves trust, it extends beyond that singular concept. Mere trust does not ensure that the collaboration between two entities will yield outcomes surpassing their separate contributions. Some barriers stand in the way of achieving synergy between software practitioners and AI4SE solutions, including the following:

Piscem natare doces: In the survey of Microsoft practitioners mentioned earlier [44], several participants pointed out that they deemed certain research unnecessary, as the resultant solutions were not seen as essential. This sentiment was particularly more pronounced among experts; the study found that as experience increased, participants were more critical and considered more studies as unimportant as well as fewer studies as essential. This observation was statistically significant with a p-value of 0.01.

Disrupting the "flow": Software practitioners are most effective when they are in a state of "flow", a concept described as "a state in which people are so involved in an activity that nothing else seems to matter" [59]. This state has been proven crucial for the productivity of software practitioners [60], [61]. If AI4SE solutions are introduced inappropriately or at inopportune moments, software practitioners may feel disrupted, much like the annoyance users felt with Microsoft's Clippy, which was deemed "annoying, impolite, and disruptive of a user's workflow" [62].

Resistance to change: People often display an aversion to modifying their established routines. Previous research has highlighted software practitioners' hesitancy to embrace new processes [63] or technologies [64]. Merely introducing an AI4SE solution does not ensure immediate adoption and endorsement, especially if it disrupts familiar practices. Such resistance can arise if AI4SE solutions are not smoothly integrated into the technological environments software practitioners are accustomed to. For instance, many AI4SE solutions have not been integrated into popular IDEs or issue-tracking systems, hampering their adoption. Another possible situation where resistance may happen is when employing an AI4SE solution may necessitate practitioners to adopt new procedures. For example, to leverage a fault localization solution, practitioners are prompted to consult a ranked list of potentially buggy program locations – a step they may not practice before the introduction of such a solution. The need for a change in *modus operandi* may pose a certain resistance that needs to be effectively managed (c.f., [65]).

Differences in abstraction levels: Practitioners often consider overarching goals, which may involve a workflow of many tasks, each further broken down into micro-tasks. For example, the DevOps workflow consists of multiple phases, and each phase, e.g., coding, includes multiple activities, e.g., navigation, editing, comprehension, etc. [66], [67]. In contrast, current AI4SE solutions usually target specific, narrower micro-tasks, such as fault localization, clone detection, API recommendation, code summarization, duplicate bug report detection, etc. While each of these micro-tasks is important, the lack of understanding of the overarching workflow may be a barrier to effective synergy.

Communication barriers: The way humans communicate with each other when collaborating on tasks differs from human-AI4SE solution interactions. While humans have a wide range of communication means to collaborate with each other – text, code, sketches [68], and so forth – their communication means with AI4SE solutions are much more limited. Moreover, humans engage in multi-round exchanges [69], drawing from both short [70] and long-term [71] memories of past interactions. Many AI4SE solutions, on the other hand, operate in a single interaction mode. For instance, in fault localization [53], [54], most solutions simply allow practitioners to provide a set of program spectra (corresponding to failing and successful test cases). The AI4SE solution then returns a list of potential faulty program locations. A similar observation can be made for many other AI4SE solutions, e.g., code search [72], code summarization [73], etc.

These challenges can hinder effective synergy, making it difficult for practitioners to fully benefit from AI4SE solutions.

III. Vision: Software Engineering 2.0

"*Anything one can imagine, others can make real*"
– Jules Verne

What possibilities can a trustworthy and synergistic AI4SE unveil? This section paints a future shaped by trustworthy and synergistic AI4SE. Different from conventional papers that chronicle past achievements, this section chooses instead to spotlight the potential of what can be, given the necessary leaps in innovation. With trustworthy and synergistic AI4SE maturing, we are steadily advancing toward establishing a *symbiotic partnership between software practitioners and autonomous, responsible, and intelligent AI4SE agents, creating a human-AI hybrid workforce*. The realization of this human-

AI hybrid workforce heralds a new era of **Software Engineering 2.0 (SE 2.0)**.[5]

Current state

Over the past two decades of research in AI4SE, we have witnessed the rise of AI4SE tools. As shown in Fig. 2(a), platforms like GitHub Copilot, Amazon CodeWhisperer, and OpenAI ChatGPT, are now embraced by many professional and aspiring software practitioners. This represents a big shift from AI4SE's nascent stages when AI capabilities were limited, drawing insights only from much smaller datasets for binary predictions (such as defect prediction [75] and failure prediction [76]) or generating contents confined to a strict form or grammar (such as specification mining [7]).

However, to actualize the vision of Software Engineering 2.0, there remain significant challenges to address. SE 2.0 envisions a reality that transcends merely equipping developers with AI tools for rudimentary tasks—like generating standard boilerplate code or commonplace algorithms; tasks that today's tools have already mastered. Rather, it gestures toward a broader, more transformative horizon.

From smart tools to smart workmates

As research in trustworthy and synergistic AI4SE advances, AI4SE tools will evolve from mere smart tools to smart workmates — see Fig. 2(b). The defining trait that distinguishes a smart workmate from a smart tool is *responsible autonomy* – something we expect from a human colleague in software development. When a tool is trustworthy and can synergize well with practitioners, it can operate with increased *autonomy*, with practitioners confident in its ability to *responsibly* execute tasks. This is a big transition analogous to a transformation from a smartwatch to a colleague, reminiscent of the android Data from the Star Trek series.

Moreover, currently, AI4SE tools predominantly function as assistants. As these tools mature further, they stand to be recognized as *first-class citizens* within software development processes. As first-class citizens, these tools will evolve into smart workmates that can assume a broader array of roles. They can act as peers, aiding in the development of software modules with limited supervision, or even adopt managerial capacities, performing work planning and coordination tasks – see Fig. 2(c).

Furthermore, AI4SE workmates, acting as intelligent agents, will not be limited to collaborating with just one individual. Instead, they will become integral members of teams. Picture a blended team of software practitioners and AI4SE intelligent agents working cohesively toward shared objectives as illustrated in Fig. 2(d). This setup will involve diverse interactions: human-to-human (H-H), human-to-agent (H-A), and agent-to-agent (A-A). While substantial research exists on H-H and H-A dynamics, the A-A interactions are much less explored, especially in the software engineering field. Central to these interactions is the establishment of symbiosis and synergy, enabling mutual enrichment between practitioners and AI4SE agents.
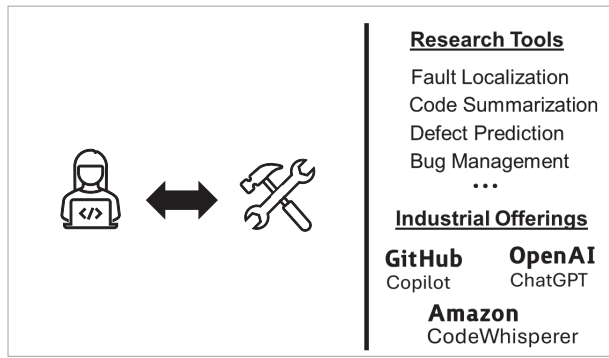
Adaptable yet solid

Today's software practitioners navigate a dynamic environment. Team members come in from the job market and, in time, move on. In the envisioned SE 2.0, AI4SE agents must be agile and adaptable to effortlessly integrate into teams as illustrated in Fig. 2(e). They should be capable of discerning the strengths and capabilities of both human members and fellow AI4SE agents, identifying avenues to contribute meaningfully to the team's goals. Moreover, these agents should possess the resilience and flexibility to adjust when either their AI counterparts or software practitioners transition out of the team.

Lastly, the viability of SE 2.0 hinges on solid legal, ethical, and economic foundations as illustrated in Fig. 2(f). There may be a need for new legal frameworks to delineate responsibility when software practitioners and intelligent agents collaborate. Privacy and copyright regulations will likely require adjustments to cater to SE 2.0 dynamics. Ethical concerns must be addressed to ensure that integrating AI4SE agents into software engineering processes yields societal benefits while mitigating potential adverse impacts, such as job losses for software practitioners. From an economic standpoint, aspects like the AI4SE agent market dynamics and vendor profitability models need attention. Therefore, the evolution and implementation of SE 2.0 will necessitate contributions not only from the Software Engineering field and Computer Science discipline but also from broader academic and professional domains.
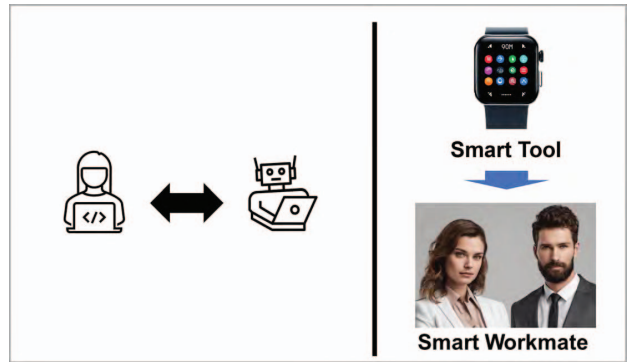
Timeline

During the session in which this talk was presented at ICSE 2023, an engaging discussion emerged about the timeline for SE 2.0's realization. The transition from the current SE to SE 2.0 will unfold in phases. Currently, we observe a surge of enthusiasm among software practitioners to harness AI4SE tools. However, there are mixed results and many unresolved issues; current tools are often cumbersome and ineffective for various software engineering tasks. They are also only able to automate some of the many tasks that software practitioners do today.
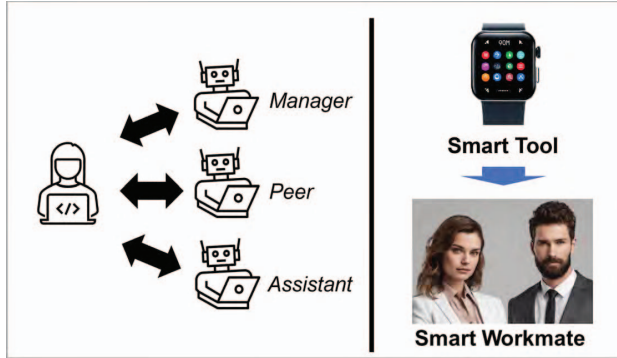
In the upcoming phase – `Now` to `Now+U` years – buoyed by increased AI4SE research and substantial investments from academia, industry, and government, many of the challenges mentioned in the previous paragraph will be addressed. This will transform current AI4SE tools into "power" tools. These "power" tools will still not be autonomous and require practitioners' close supervision. However, they will address and

---

[5]Software Engineering 2.0 is distinct from the concept of Software 2.0. Software 2.0 is defined as software that "is written in much more abstract, human unfriendly language, such as the weights of a neural network. No human is involved in writing this code." [74] Software Engineering 2.0 focuses on constructing autonomous, responsible, intelligent AI4SE agents that can symbiotically work with software practitioners to collaboratively build software, whether they are Software 1.0, Software 2.0, or possibly future Software X.0. Given the unique advantages and limitations of both Software 1.0 and Software 2.0, it is anticipated that future software systems will integrate both, thereby creating complex composite software systems where the need for Software Engineering 2.0 will be even more significant.
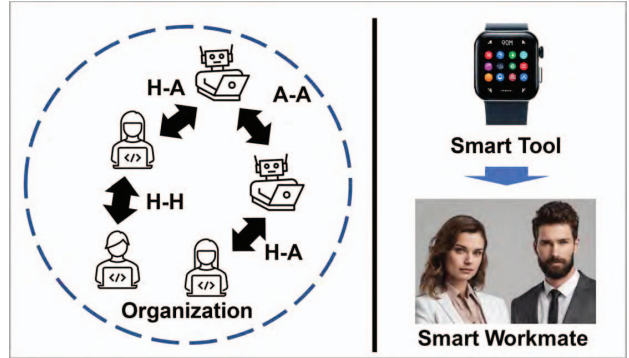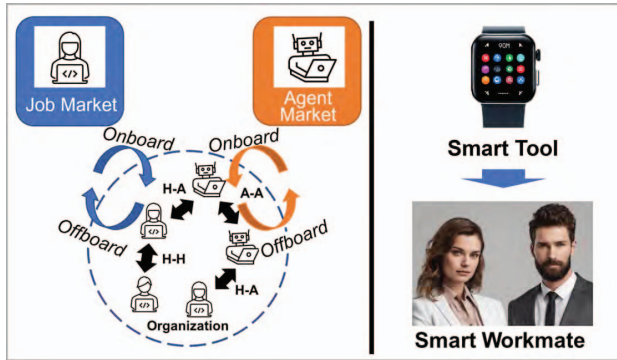
(a) Current state

(b) Smart tool to workmate, exercising *responsible autonomy*
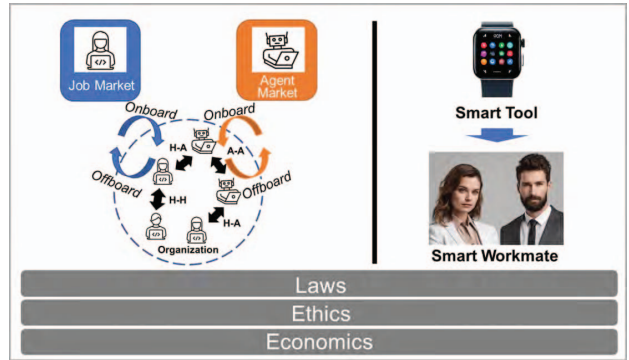
(c) First-class citizens, taking different roles

(d) Able to work in a team with diverse interactions

(e) Able to deal with dynamic environment

(f) Require solid foundations in multiple disciplines

Fig. 2: Software Engineering 2.0 (SE 2.0) will see AI4SE solutions transitioning from smart tools to smart workmates. These intelligent agents will exercise responsible autonomy. They will be first-class citizens, taking different roles as assistants, peers, and even managers in software engineering projects. They will be well integrated in dynamic software engineering teams. There will be human-human (H-H), human-agent (H-A), and agent-agent (A-A) interactions. There will also be a new economy surrounding the AI4SE agent market as well as solid foundations in law and ethics supporting SE 2.0.

alleviate many of the frustrations practitioners currently face in using them, and be integrated smoothly across a broad range of software engineering tasks. Drawing a parallel, consider the progression of smartwatches. Their genesis can be traced back to 1976, when they were expensive, offered limited functionality like basic calculations, and had many usability issues. In contrast, the smartwatches of today, nearly half a century later, are versatile and user-friendly gadgets. Given today's accelerated pace of innovation, it is plausible that

the evolution of AI4SE tools will occur in a considerably shortened timeframe, although predicting a precise value for `U` is challenging.

In the subsequent phase – `Now+U` years to `Now+(w×U)` years – these "power" tools will evolve into intelligent workmates, characterized by responsible autonomy. This progression will likely be incremental too, unfolding as the facets of responsible autonomy depicted in Fig. 2(b) – (f) are actualized and their associated challenges addressed. It is at this juncture

that Software Engineering 2.0 will truly come into being. The value for w will depend on how fast Artificial General Intelligence (AGI) will be realized. Projections vary significantly, with some anticipating some form of AGI realization in a few years, while others expect it to take several decades [77], [78].

IV. ROADMAP TO TRUSTWORTHY AI4SE

"*Trust but verify*" – Ronald Reagan

Trust is a pivotal element in successful collaborations between humans and intelligent solutions [79]–[81]. Section II has highlighted the challenges in establishing trust between software practitioners and AI4SE solutions. Thus, further research is important. This section outlines nine strategies for achieving trustworthy AI4SE, as depicted in Fig. 3 and detailed further below.

T1 *Characterize trust factors*

Firstly, a clearer definition is required regarding the factors that influence practitioners' trust in AI4SE solutions. Some studies have looked into measuring trust in automation, e.g., [82], [83], however, mostly not in the context of software engineering. While some studies have delved into practitioners' expectations of specific AI4SE solutions [45], [84]–[86], the scope remains limited. A broader examination that covers many more AI4SE solutions is needed. Also, although expectation is related to trust, they are not synonymous. The distinctions between the two warrant explorations in future empirical research.

A wide-ranging empirical study, encompassing a diverse group of software practitioners considering diverse AI4SE solutions, will aid in comprehensively understanding trust factors influencing software practitioners in working with AI4SE solutions. An exciting recent work is by Johnson et al. who interviewed 18 practitioners within and outside Microsoft and uncovered factors that influence practitioners' trust in software tools, including AI4SE ones [87]. There is a need for more studies in this direction to corroborate and expand upon these factors, as well as to uncover potentially context-dependent factors that align with specific software engineering tasks and characteristics of specific AI4SE solutions.

Furthermore, it is worth noting that practitioners' perceptions of trust factors might not be entirely accurate (c.f., [88]). Therefore, a multifaceted research approach encompassing interviews, surveys, controlled experiments, and field studies is crucial. This holistic effort will pave the way to comprehensively understand "trust" within the AI4SE context.

T2 *Prevent misplaced trust and calibrate trust*

Trust in automation is a nuanced matter; when rightly placed, it accelerates productivity; but when misplaced, it leads to detrimental consequences. Prior studies have highlighted the dangers of automation misuse, e.g., [79]. These concerns are also applicable to AI4SE which increasingly offers more and more automation. Despite the advances in AI4SE, there are still frequent failures. Over time, even as AI4SE solutions

become increasingly sophisticated and get closer to passing the Turing test, they are likely to still be susceptible to errors, mirroring the adage "to err is human."

Complicating this is the propensity of AI4SE solutions, especially those that are based on Large Language Models (LLMs), to yield outputs that appear fluent or accurate at first glance, but contain issues or subtle inaccuracies. To illustrate, a recent study indicated that while ChatGPT-generated answers to software engineering technical questions are almost as fluent as those of humans, they are often much less useful [89]. More concerning is the risk of AI4SE solutions generating wrong or harmful outputs due to hallucination [46], [47]. This necessitates the development of technologies that can identify the aforementioned issues, including hallucination, to prevent unwarranted trust.

Additionally, there is a need for ongoing, unbiased evaluation of AI4SE solution capabilities, especially given their rapid evolution. This ensures that both practicing and budding software practitioners can judiciously determine their trust levels. There is also a need for the design of mechanisms that allow for audit and accreditation of AI4SE solutions, either with or without a trusted third party, c.f., [90], [91].

Improve trustworthiness of AI4SE solutions

Currently, without an exhaustive empirical study, it is challenging to pinpoint all the factors that delineate or relate to trust for diverse software engineering tasks. However, one can infer that a software practitioner's trust in an AI4SE solution might be influenced by factors that fall into two dimensions: intrinsic and extrinsic. Intrinsic factors relate to an AI4SE solution's ability to produce accurate outputs and provide explanations for its results, while extrinsic factors concern with, for example, the AI4SE solution's adherence to external authority regulations and its resilience against external threats. Sections IV-A and IV-B discuss some strategies to enhance the trustworthiness of AI4SE solutions in these two dimensions.

A. Enhance Intrinsic Trustworthiness of AI4SE Solutions

Naturally, software practitioners tend to place greater trust in AI4SE solutions that consistently deliver accurate results for assigned tasks [45], [58]. Software practitioners also appreciate explanations in addition to recommendations, especially when working with AI4SE solutions [45]. To cultivate AI4SE solutions that earn developers' trust, the following directions that can boost the efficacy of AI4SE solutions and improve their ability to provide explanations will be worth pursuing.

T3 *Design trust-aware efficacy metrics*

Evaluating the trustworthiness of AI4SE solutions demands efficacy metrics that mirror factors influencing practitioner trust. For example, several studies have highlighted top-N accuracy as a good efficacy metric for an AI4SE solution that produces a ranked list of recommendations [45], [55]. This metric recognizes that practitioners often focus only on the top N recommendations, with N being a small number. If an AI4SE solution produces many good recommendations beyond
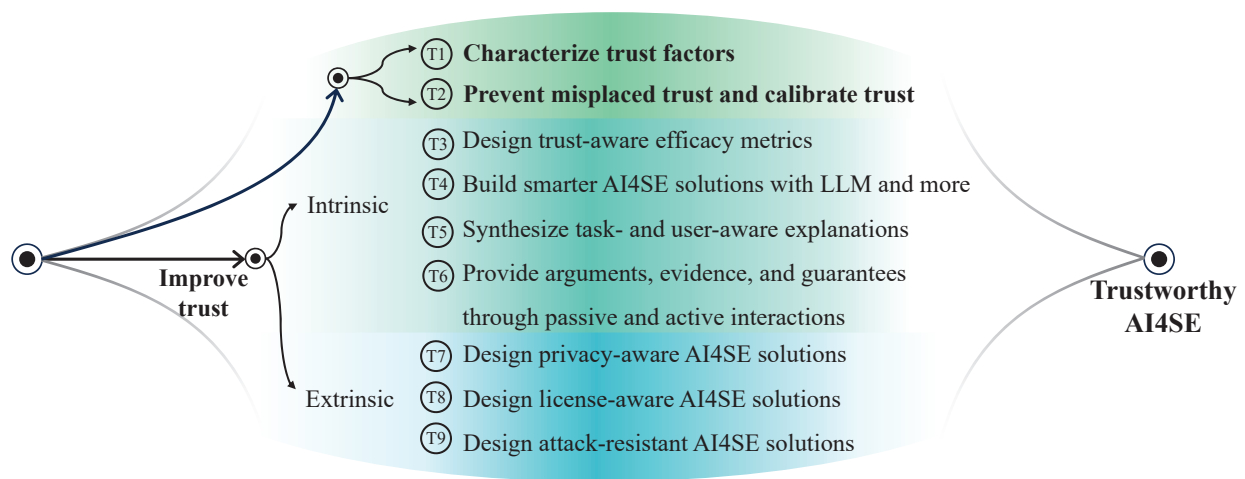
Fig. 3: Roadmap to Trustworthy AI4SE

the top-N, these do not positively influence practitioners' trust in the solution. Similarly, another study [92] has proposed the number of initial false alarms (IFA) as a metric, as first impressions matter to get a software practitioner to trust an AI4SE solution. Also, in areas like defect prediction, effort-aware metrics have been proposed [93]. These metrics are more aligned with trust factors compared to their non-effort-aware counterparts. This is because practitioners may find their trust dwindling if they do not witness an increase in favorable outcomes that is commensurate with an increased effort in scrutinizing an AI4SE solution's recommendations.

(T4) *Build smarter AI4SE solutions with LLM and more*

The area of AI4SE offers much potential for advancement. At present, a prominent research trend within the AI4SE community is the use, adaptation, and design of Large Language Models (LLMs) to automate software engineering tasks. Current studies predominantly harness these LLMs for a subset of software engineering tasks, such as code search and code summarization [94], [95]. However, software engineering is multifaceted, encompassing more than just these tasks. While recent research has begun delving into less-explored areas, e.g., managing bug reports [96], designing software architecture [97], building navigation aids for software engineering Q&A sites [98], etc., there remains significant scope for broader exploration.

Additionally, recent studies have highlighted certain limitations of LLM for software engineering (LLM4SE). For instance, these models are not robust to minor perturbations of inputs [99], [100]. Also, they can be vulnerable to shifts in data distributions, like the evolution of third-party libraries and releases of new ones [101]. Moreover, a prior study has also demonstrated the challenges of LLM4SE in handling data that resides in the tail-end of the distribution [102]. Recognizing and characterizing the limitations of LLMs and devising strategies to overcome those presents compelling directions for future research.

To boost the efficacy of LLM4SE, some recent studies have focused on enhancing the inputs and outputs of LLMs and integrating them with other techniques. For instance, some studies incorporate code graphs (e.g., control flow graph, program dependency graph, etc.) as input to LLMs [72], [103]. Others transform source code to an intermediate representation (IR) that provides a more concise and uniform input to LLMs for more effective and efficient learning [104], [105]. Moreover, other studies emphasize selecting optimal in-context examples to bolster the efficacy of LLMs [106]. There is also an emphasis on coupling LLMs with methods like program analysis and testing to yield superior results [47], [107]–[111]. Additionally, a few studies showcase the potential of both one-off and ongoing interactions with LLMs to refine their outputs for several software engineering tasks [47], [112].

Owing to the swift progress in LLM4SE research, this concise summary does not capture its entire scope and may be quickly outdated. As such, Systematic Literature Reviews (SLRs) on LLM4SE, e.g., [42], can prove invaluable. The rapid evolution in LLM4SE warrants multiple SLRs to capture the latest developments and trends. Multiple SLRs can also be conducted to examine LLM4SE from various perspectives.

(T5) *Synthesize task- and user-aware explanations*

Contemporary AI4SE solutions offer diverse recommendations – including patches to fix a bug, source code to write next, third-party library to use, and so on. However, these often come without explanations. This opacity can diminish the trust software practitioners have in these suggestions. As a result, there is a need for explainable AI4SE solutions that can realize effective practitioner-AI4SE solution interactions that engender trust and bring about trusted collective intelligence.

While there are some efforts done in this direction, e.g., [113]–[117], more explorations are needed. First, we can extend existing explainable AI4SE solutions to cover more software engineering tasks. Second, we can develop their capabilities to produce *task-specific explanations* that

are tailored to suit specific software engineering tasks and contexts. Moreover, the explanations need to be *user-aware*; they need to consider the specific expertise and experience of a practitioner who uses an AI4SE solution. Producing effective explanations for a recommendation made by an AI4SE solution given a specific task and context for a target software practitioner is challenging for several reasons: software engineering is a complex endeavor including diverse tasks; software engineering knowledge evolves rapidly; also, software practitioners have diverse backgrounds. Thus, there is much room to innovate to tackle these challenges.

(T6) *Produce arguments, evidence, and guarantees through passive and active interactions*

For greater acceptance by software practitioners, AI4SE solutions must articulate convincing arguments and present pertinent evidence to practitioners, guiding their decisions regarding recommendations made by AI4SE solutions. Furthermore, evidence should not be solely obtained passively from the data present in software artifacts. Active interaction with these artifacts – feeding input data and monitoring the ensuing outputs – can be an effective way of producing evidence. Additionally, it will be desirable if AI4SE solutions are able to provide some guarantees of their efficacy.

Consider an instance where an AI4SE solution suggests to a practitioner that a particular code segment is potentially vulnerable. Its *argument*: the code resembles M code fragments on GitHub, which experienced practitioners fixed to remove vulnerabilities that match an entry in the Common Weakness Enumeration (CWE).[6] To bolster this argument, the AI4SE solution can produce tangible *evidence* by generating a test input that showcases the exploitability of the flagged code, following techniques proposed in [118]–[120].

Similarly, an AI4SE solution recommending third-party libraries [121] may execute static and dynamic program analyses. These will result in concrete pieces of evidence, e.g., demonstrable efficiency or a small memory footprint, to support the recommendations. A worst-case execution time guarantee can also be given by leveraging static analysis [122].

B. Enhance Extrinsic Trustworthiness of AI4SE Solutions

Software practitioners need guarantees that the use of AI4SE solutions will not bring them into conflict with laws. Also, they need assurances that sufficient measures are in place to protect AI4SE solutions from malicious actors. To address these concerns, which impact the trustworthiness of AI4SE solutions, it is essential to explore the following directions.

(T7) *Design privacy-aware AI4SE solutions*

An open avenue of research is in fortifying AI4SE solutions to ensure compliance with privacy regulations more rigorously. Prior works have provided some protection to sensitive data in some software artifacts [123]–[125]. They, however, have mainly focused on data that comes in tabular format. They

[6]https://cwe.mitre.org/index.html

have also only considered some specific software engineering tasks, e.g., testing, defect prediction, and debugging. Thus, more can be done to strengthen this protection and expand it to cover diverse software artifacts and tasks.

Also, privacy-aware AI4SE solutions are important for the adoption of AI4SE solutions that require software practitioners to transmit proprietary code and data to third-party services. Software practitioners (and companies) need certain guarantees that proprietary code and data are not retained or seeped into the underlying AI4SE model that can potentially be leaked to other users of the service, c.f., [126], [127]. The use of Trusted Execution Environments [128] and the design of suitable protocols may be one way to achieve this guarantee, c.f., [91]. Alternatively, model compression techniques can be used to customize AI4SE models for local deployment on servers with limited memory and processing power [129]. Such local deployment eliminates the need to transfer proprietary data to third-party vendors, improving privacy. Yet another possibility is to design AI4SE solutions that employ federated learning [130].

Moreover, strategies are needed to help AI4SE solutions respect the EU General Data Protection Regulation (GDPR)'s provision of "right to be forgotten" without necessitating extensive retraining, c.f., [131]. The current AI4SE solutions do not readily allow the removal of specific contributions from open-source projects when the corresponding software practitioners invoke their "right to be forgotten" under GDPR.

(T8) *Design license-aware AI4SE solutions*

In 2022, Microsoft, GitHub, and OpenAI were sued for issues related to privacy and copyright [132]. Moreover, an AI-powered coding assistant can inadvertently replicate GPL v3 licensed code from GitHub, risking copyright infringements when integrated into proprietary software. Holistic integration of licensing information and constraints [133] into AI4SE solutions' training or fine-tuning processes can be designed to address the aforementioned problems. Concurrently, runtime checking methods can be developed to flag AI4SE recommendations that potentially infringe on licensing terms. Code clone detection methods, e.g., [134]–[137], can be employed as part of such runtime checks.

(T9) *Design attack-resistant AI4SE solutions*

As AI4SE solutions become integral to software engineering processes, their ability to withstand attacks from malicious actors is of high importance. Recent studies [99], [138]–[143] have examined specific attack strategies, yet a broader spectrum awaits exploration. A comprehensive threat assessment is vital, paired with detection, quantification, and mitigation strategies, aiming to design and develop AI4SE solutions that are resilient to multifaceted attacks. Strengthening AI4SE defenses, devising corrective algorithms for real-time "self-healing", and probing the ramifications of data poisoning on software artifacts should be prioritized. Some strides have been made in this direction, but more is warranted.

V. ROADMAP TO SYNERGISTIC AI4SE

"The whole is greater than the sum of its parts" – Aristotle

Section II highlighted a number of barriers that impede synergistic interactions between software practitioners and AI4SE solutions. Thus, further research is essential. This section outlines six strategies for achieving synergistic AI4SE, as illustrated in Fig. 4 and elaborated upon below.

A. Understand synergy

(S1) *Characterize strengths and weaknesses for better practitioner-AI4SE solution synergy*

To ensure that collaborations between software practitioners and AI4SE solutions produce outcomes greater than the sum of their individual contributions, it is essential to recognize the strengths of one party that can offset the weaknesses of the other. In software engineering, tasks often form lengthy chains, with each main task comprising several micro-tasks. Software practitioners may excel in certain micro-tasks but falter in others. Similarly, AI4SE solutions may perform very well in certain micro-tasks but perform poorly in others. The sets of micro-tasks that software practitioners and AI4SE solutions excel in may be substantially different. Hence, there is a need for empirical research to determine which micro-tasks are better assigned to practitioners – bearing in mind that the aptitude for micro-tasks can differ among individuals and can evolve with their experience. Similarly, it is important to identify the tasks best assigned to AI4SE solutions, acknowledging that these too can vary based on the capabilities of specific AI4SE solutions and the characteristics of practitioners using them.

Also, not in all situations AI4SE solutions can help. To underscore this point, a past study at Adobe revealed that junior software practitioners have a lower adoption threshold for a bug localization solution[7] [144]. On the other hand, experienced practitioners indicated less need for such automated assistance and have a higher threshold for adoption. Therefore, for AI4SE bug localization solutions to genuinely complement these experienced software practitioners, they need to surpass a certain level of efficacy. Failing to do so, these solutions might prove more obstructive than beneficial. One possibility is for such solutions to provide recommendations sparingly only in cases for which they are likely to exceed the experienced practitioners' expectations, c.f., [145]–[147].

As AI4SE matures and AI4SE solutions become first-class citizens and autonomous agents, synergizing software practitioners and AI4SE agents can be conceptualized as an optimization problem. Specifically, assuming that we can quantitatively characterize the strengths and weaknesses of different software practitioners and AI4SE solutions for different tasks as weights, the problem of synergizing software practitioners and AI4SE solutions to achieve results that best

[7]A bug localization solution produces a ranked list of potentially buggy source code files given a bug report.

exceed the sum of their individual contributions is a task assignment problem. In essence, we need to assign tasks to the most suitable software practitioners and/or AI4SE agents to maximize overall reward or efficacy.

(S2) *Understand flow and reimagine processes*

Software practitioners occasionally need assistance, but not incessantly. Ill-timed assistance can be counterproductive and break software practitioners' "flow." While we can allow practitioners to manually toggle assistance on and off, they may not always discern the optimal moments for aid. Consequently, further studies are vital to pinpoint when AI4SE solutions are most beneficial, integrate them smoothly into the software development process to enhance software practitioners' "flow", tailor them to individual practitioners with varied preferences, and measure the benefits they bring to the table.

Also, many of the software engineering processes today do not prominently picture AI4SE solutions. As AI4SE solutions transition from tools to smart workmates, we may need to re-look into the existing processes and identify limitations that may impede synergy. There may be a need for new processes that facilitate symbiotic partnerships between practitioners and AI4SE agents. In such a partnership, both entities mutually benefit, working toward shared goals more effectively.

(S3) *Understand change process toward synergy*

Satir change model [148] delineates several stages that emerge when a change is introduced: previous status quo, resistance, chaos, integration, and new status quo. This model has been influential in organizational behavior and even in the adoption of new software engineering methodologies. For instance, Lindstrom and Beck noted that following the Satir change model, adoption of eXtreme Programming does not guarantee instant positive outcomes and benefits may appear gradually [149].

Similarly, the introduction of AI4SE solutions may require individuals to undergo a change process to achieve synergy. This change process may not instantly lead to improved outcomes. Departing from a familiar status quo, the introduction of an AI4SE solution – which is perceived as an external element – can incite resistance and even chaos. But this phase can subsequently give way to transformation and integration, ultimately establishing a new status quo.

In AI4SE research, there have been limited studies on this change process. There is a need for such studies to answer questions such as: How can we measure the efficacy of the change process? What strategies can accelerate the change process, allowing us to reach the new status quo more rapidly? How can we increase the likelihood that the new status quo significantly surpasses the previous one in terms of quality (of the software developed) and productivity (of software practitioners)? And lastly, what automated solutions can be developed to alleviate the challenges software practitioners face during this transformation?
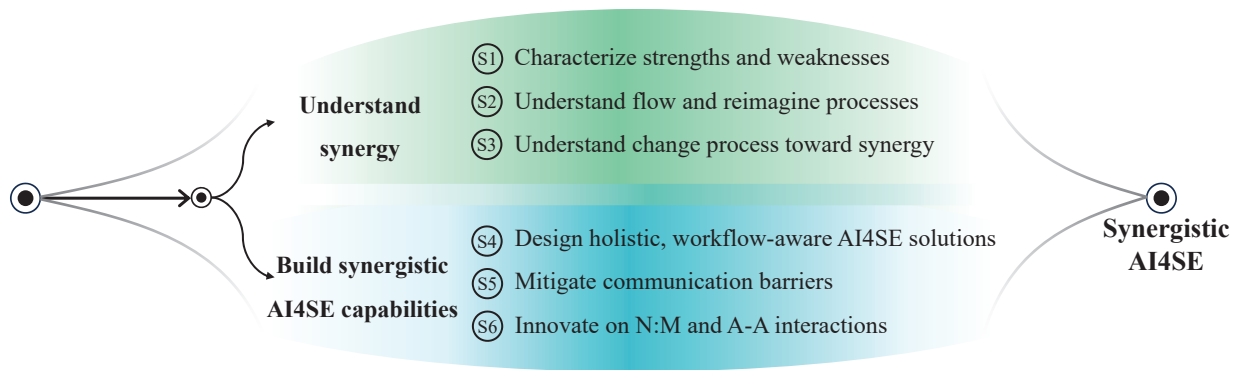
Fig. 4: Roadmap to Synergistic AI4SE

B. Build synergistic AI4SE capabilities

(S4) *Design holistic and workflow-aware AI4SE solutions*

Most AI4SE solutions are tailored for specific micro-tasks, such as code generation, code summarization, fault localization, or duplicate bug report detection. In contrast, as highlighted in Section II, software practitioners often operate at a different level of abstraction, considering higher-level and broader objectives. This difference may be a barrier to synergy.

To address this challenge, a shift toward holistic, workflow-aware AI4SE solutions can be beneficial. For example, as a start, AI4SE solutions can transfer information and insights from one micro-task to others, c.f. [150], [151]. Additionally, they can leverage practitioners' interactions and feedback in one micro-task to improve their capabilities in others. Moreover, we can build AI4SE solutions that pivot from assisting isolated micro-tasks to improving the entire workflow, ensuring that they capture the broader picture rather than just the individual micro-tasks – seeing the "forest" instead of "trees".

(S5) *Mitigate communication barriers*

Many AI4SE solutions currently offer limited interactivity. Though solutions built on top of ChatGPT have begun bridging these interaction gaps, they still fall short of replicating the deeply collaborative experience akin to pair programming with a trusted colleague. Also, in spite of the strides made by LLM-powered AI4SE solutions, challenges persist, such as interactions that become stagnant or unproductive toward solving a software engineering task [47].

To truly capitalize on the potential of AI4SE solutions, we must enhance the communication capabilities between software practitioners and AI4SE solutions. The ideal scenario will allow practitioners to interact with AI4SE solutions as naturally and effectively as they do with their peers. Achieving seamless communication is pivotal for fostering synergy. Moreover, the means of communication should extend beyond just text and code to encompass visuals (like diagrams and sketches), gestures (captured via videos or wearables), and other modes of interaction. Advances in Foundation Models that go beyond LLMs and consider modalities such as images

and videos, e.g., [152], may pave the way toward better communication between practitioners and AI4SE solutions.

(S6) *Innovate on N:M and A-A interactions*

Most existing AI4SE solutions focus on the `1:1` usage scenario, wherein a single software practitioner engages with one AI4SE solution. To more fully unlock the potential of AI4SE, we need to explore the `N:M` usage scenario, wherein several software practitioners synergistically interact with multiple AI4SE solutions in a team. Research on this `N:M` usage scenario is limited. Considering that most software engineering teams consist of more than one practitioner, it is important to explore how a team of interacting practitioners and AI4SE solutions can collaboratively and synergistically complete tasks.

As AI4SE evolves to take a central role in software engineering, it becomes imperative to investigate the synergy among AI4SE solutions. Currently, limited research delves into how these solutions can recognize the capabilities of their counterparts, distribute tasks, and collaboratively operate by capitalizing on each other's strengths and addressing each other's weaknesses. These collaborative capabilities should be developed for both static (where collaborations occur among a predefined set of AI4SE solutions) and dynamic (where collaborations involve previously unknown AI4SE solutions) settings.

VI. SUMMARY AND CONCLUDING REMARKS

"If you want to go far, go together" – African Proverb

The area of AI for Software Engineering (AI4SE) has witnessed exponential growth over the past two decades. Evolving from a niche segment centered around a handful of software engineering tasks, AI4SE has grown into a key pillar within the software engineering field, underscored by the prowess of specialized AI4SE solutions across a wide range of software engineering tasks. Three distinct waves of innovation have shaped the trajectory of AI4SE: the surge of software engineering big data, the incorporation of deep learning into the design of AI4SE solutions, and, more recently, the development of AI4SE solutions based on large language models.

The latter has propelled AI4SE into the limelight, spawning industry-grade solutions that have found widespread adoption.

This paper highlights two key challenges still confronting AI4SE: the need for trust and synergy. These intertwined principles are important for harnessing the full potential of AI4SE. Looking forward, trustworthy and synergistic AI4SE solutions can transform our present AI4SE tools into truly intelligent workmates, characterized by *responsible autonomy*. These AI4SE entities, acting as intelligent agents, will seamlessly integrate as key autonomous contributors, performing varied roles adeptly and responsibly in software engineering workflows. Moreover, their adaptability will empower them to operate symbiotically not just alongside individuals but within larger, dynamic teams where members – both humans and intelligent agents – can dynamically change. This transformation, fueled by the progression of AI to Artificial General Intelligence (AGI), will mark the inception of Software Engineering 2.0. While advancements in software engineering and AI research are clearly important, the successful realization of the Software Engineering 2.0 vision also hinges on strong foundational frameworks in the legal, ethical, and economic domains.

To achieve the aforementioned vision of Software Engineering 2.0, this paper presents two roadmaps steering toward trustworthy and synergistic AI4SE. The roadmaps enumerate 15 open challenges that await further attention of the AI4SE community. Addressing these could move AI4SE ever closer to Software Engineering 2.0.

This paper seeks to motivate more to join and contribute to the AI4SE research journey. AI4SE is currently in a "*Belle Époque*" mirroring the aeronautics revolution of the early 1900s. Just as the Wright brothers' groundbreaking 12-second flight in 1903 redefined the boundaries of travel, the promising advances of AI4SE hold the potential to reshape the future of software engineering. Similar to aviation post its inaugural flight, sustained collaboration and dedication over many years by numerous contributors is pivotal to fully realizing the power of AI4SE in revolutionizing the software engineering landscape.

Acknowledgments

References

[1] S. C. Johnson, "Lint, a C program checker," *Computing Science TR*, vol. 65, 1977.

[2] E. M. Clarke and E. A. Emerson, "Design and synthesis of synchronization skeletons using branching-time temporal logic," in *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, ser. Lecture Notes in Computer Science, D. Kozen, Ed., vol. 131. Springer, 1981, pp. 52–71.

[3] J. Queille and J. Sifakis, "Specification and verification of concurrent systems in CESAR," in *International Symposium on Programming, 5th Colloquium, Torino, Italy, April 6-8, 1982, Proceedings*, ser. Lecture Notes in Computer Science, M. Dezani-Ciancaglini and U. Montanari, Eds., vol. 137. Springer, 1982, pp. 337–351.

[4] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of UNIX utilities," *Commun. ACM*, vol. 33, no. 12, pp. 32–44, 1990.

[5] Y. Bengio, "Springtime for AI: The rise of deep learning," *Scientific American*, 2016, accessed: 2023-08-30. [Online]. Available: https://www.scientificamerican.com/article/springtime-for-ai-the-rise-of-deep-learning/

[6] T. Xie, S. Thummalapenta, D. Lo, and C. Liu, "Data mining for software engineering," *Computer*, vol. 42, no. 8, pp. 55–62, 2009.

---

[8]https://www.bing.com/create

[9]https://dreamstudio.ai

[10]https://www.acm.org/publications/policies/frequently-asked-questions

[7] D. Lo and S. Khoo, "SMArTIC: towards building an accurate, robust and scalable specification miner," in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2006, Portland, Oregon, USA, November 5-11, 2006*, M. Young and P. T. Devanbu, Eds. ACM, 2006, pp. 265–275.

[8] H. Cheng, D. Lo, Y. Zhou, X. Wang, and X. Yan, "Identifying bug signatures using discriminative graph mining," in *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis, ISSTA 2009, Chicago, IL, USA, July 19-23, 2009*, G. Rothermel and L. K. Dillon, Eds. ACM, 2009, pp. 141–152.

[9] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, W. Schäfer, M. B. Dwyer, and V. Gruhn, Eds. ACM, 2008, pp. 461–470.

[10] D. W. Binkley and D. J. Lawrie, "Development: Information retrieval applications," in *Encyclopedia of Software Engineering*, P. A. Laplante, Ed. Taylor & Francis, 2010, pp. 231–242.

[11] M. P. Robillard, E. Bodden, D. Kawrykow, M. Mezini, and T. Ratchford, "Automated API property inference techniques," *IEEE Trans. Software Eng.*, vol. 39, no. 5, pp. 613–637, 2013.

[12] D. Partridge, "Artificial intelligence and software engineering: A survey of possibilities," *Information and Software Technology*, vol. 30, no. 3, pp. 146–152, 1988.

[13] A. E. Hassan, R. C. Holt, and A. Mockus, "Report on MSR 2004: International workshop on mining software repositories," *ACM SIGSOFT Softw. Eng. Notes*, vol. 30, no. 1, p. 4, 2005.

[14] M. Harman, S. Poulding, and M. D. Penta, Eds., *Search Based Software Engineering - First International Symposium, SSBSE 2009*, 2009.

[15] M. Li, H. Zhang, D. Lo, and Lucia, "Improving software quality and productivity leveraging mining techniques: [summary of the second workshop on software mining, at ASE 2013]," *ACM SIGSOFT Softw. Eng. Notes*, vol. 40, no. 1, pp. 1–2, 2015.

[16] F. Thung, T. F. Bissyandé, D. Lo, and L. Jiang, "Network structure of social coding in GitHub," in *17th European Conference on Software Maintenance and Reengineering, CSMR 2013, Genova, Italy, March 5-8, 2013*, A. Cleve, F. Ricca, and M. Cerioli, Eds. IEEE Computer Society, 2013, pp. 323–326.

[17] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillère, J. Klein, and Y. L. Traon, "Got issues? who cares about it? A large scale investigation of issue trackers from GitHub," in *IEEE 24th International Symposium on Software Reliability Engineering, ISSRE 2013, Pasadena, CA, USA, November 4-7, 2013*. IEEE Computer Society, 2013, pp. 188–197.

[18] T. F. Bissyandé, F. Thung, D. Lo, L. Jiang, and L. Réveillère, "Popularity, interoperability, and impact of programming languages in 100, 000 open source projects," in *37th Annual IEEE Computer Software and Applications Conference, COMPSAC 2013, Kyoto, Japan, July 22-26, 2013*. IEEE Computer Society, 2013, pp. 303–312.

[19] X. Xia, D. Lo, X. Wang, and B. Zhou, "Tag recommendation in software information sites," in *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, T. Zimmermann, M. D. Penta, and S. Kim, Eds. IEEE Computer Society, 2013, pp. 287–296.

[20] S. Beyer and M. Pinzger, "Synonym suggestion for tags on Stack Overflow," in *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, 2015, pp. 94–103.

[21] F. Calefato, F. Lanubile, and N. Novielli, "An empirical assessment of best-answer prediction models in technical Q&A sites," *Empir. Softw. Eng.*, vol. 24, no. 2, pp. 854–901, 2019.

[22] Z. Gao, X. Xia, D. Lo, and J. Grundy, "Technical Q&A site answer recommendation via question boosting," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 1, pp. 1–34, 2020.

[23] H. Zhang, S. Wang, T.-H. Chen, Y. Zou, and A. E. Hassan, "An empirical study of obsolete answers on Stack Overflow," *IEEE Transactions on Software Engineering*, vol. 47, no. 4, pp. 850–862, 2019.

[24] H. Zhang, S. Wang, H. Li, T.-H. Chen, and A. E. Hassan, "A study of C/C++ code weaknesses on Stack Overflow," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, pp. 2359–2375, 2021.

[25] B. Xu, Z. Xing, X. Xia, and D. Lo, "AnswerBot: automated generation of answer summary to developersź technical questions," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, G. Rosu, M. D. Penta, and T. N. Nguyen, Eds. IEEE Computer Society, 2017, pp. 706–716.

[26] C. Yang, B. Xu, F. Thung, Y. Shi, T. Zhang, Z. Yang, X. Zhou, J. Shi, J. He, D. Han, and D. Lo, "Answer summarization for technical queries: Benchmark and new approach," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 2022, pp. 8:1–8:13.

[27] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE, 2015, pp. 17–26.

[28] M. White, C. Vendome, M. L. Vásquez, and D. Poshyvanyk, "Toward deep learning software repositories," in *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*, M. D. Penta, M. Pinzger, and R. Robbes, Eds. IEEE Computer Society, 2015, pp. 334–345.

[29] U. Alon, M. Zilberstein, O. Levy, and E. Yahav, "code2vec: learning distributed representations of code," *Proc. ACM Program. Lang.*, vol. 3, no. POPL, pp. 40:1–40:29, 2019.

[30] T. Hoang, H. J. Kang, D. Lo, and J. Lawall, "CC2Vec: distributed representations of code changes," in *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 518–529.

[31] Y. Yang, X. Xia, D. Lo, and J. C. Grundy, "A survey on deep learning for software engineering," *ACM Comput. Surv.*, vol. 54, no. 10s, pp. 206:1–206:73, 2022.

[32] C. Watson, N. Cooper, D. Nader-Palacio, K. Moran, and D. Poshyvanyk, "A systematic literature review on the use of deep learning in software engineering research," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 2, pp. 32:1–32:58, 2022.

[33] S. Majumder, N. Balaji, K. Brey, W. Fu, and T. Menzies, "500+ times faster than deep learning: a case study exploring faster methods for text mining stackoverflow," in *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*, A. Zaidman, Y. Kamei, and E. Hill, Eds. ACM, 2018, pp. 554–563.

[34] Z. Liu, X. Xia, A. E. Hassan, D. Lo, Z. Xing, and X. Wang, "Neural-machine-translation-based commit message generation: how far are we?" in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, M. Huchard, C. Kästner, and G. Fraser, Eds. ACM, 2018, pp. 373–384.

[35] Z. Zeng, Y. Zhang, H. Zhang, and L. Zhang, "Deep just-in-time defect prediction: how far are we?" in *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, C. Cadar and X. Zhang, Eds. ACM, 2021, pp. 427–438.

[36] H. J. Kang, T. F. Bissyandé, and D. Lo, "Assessing the generalizability of code2vec token embeddings," in *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 2019, pp. 1–12.

[37] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186.

[38] T. Zhang, B. Xu, F. Thung, S. A. Haryono, D. Lo, and L. Jiang, "Sentiment analysis for software engineering: How far can pre-trained transformer models go?" in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2020, Adelaide, Australia, September 28 - October 2, 2020*. IEEE, 2020, pp. 70–80.

[39] E. Biswas, M. E. Karabulut, L. L. Pollock, and K. Vijay-Shanker, "Achieving reliable sentiment analysis in the software engineering domain using BERT," in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2020, Adelaide, Australia, September 28 - October 2, 2020*. IEEE, 2020, pp. 162–173.

[40] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, ser. Findings of ACL, T. Cohn, Y. He, and Y. Liu, Eds., vol. EMNLP 2020. Association for Computational Linguistics, 2020, pp. 1536–1547.

[41] OpenAI, "GPT-4 technical report," *CoRR*, vol. abs/2303.08774, 2023.

[42] X. Hou, Y. Zhao, Y. Liu, Z. Yang, K. Wang, L. Li, X. Luo, D. Lo, J. Grundy, and H. Wang, "Large language models for software engineering: A systematic literature review," *CoRR*, vol. abs/2308.10620, 2023.

[43] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. T. Devanbu, "On the naturalness of software," in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, M. Glinz, G. C. Murphy, and M. Pezzè, Eds. IEEE Computer Society, 2012, pp. 837–847.

[44] D. Lo, N. Nagappan, and T. Zimmermann, "How practitioners perceive the relevance of software engineering research," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, E. D. Nitto, M. Harman, and P. Heymans, Eds. ACM, 2015, pp. 415–425.

[45] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis, ISSTA 2016, Saarbrücken, Germany, July 18-20, 2016*, A. Zeller and A. Roychoudhury, Eds. ACM, 2016, pp. 165–176.

[46] H. Pearce, B. Ahmad, B. Tan, B. Dolan-Gavitt, and R. Karri, "Asleep at the keyboard? assessing the security of GitHub Copilot's code contributions," in *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 754–768.

[47] Y. Liu, T. Le-Cong, R. Widyasari, C. Tantithamthavorn, L. Li, X. D. Le, and D. Lo, "Refining ChatGPT-generated code: Characterizing and mitigating code quality issues," *CoRR*, vol. abs/2307.12596, 2023.

[48] S. McManus. (2023) Friend or foe: Can computer coders trust ChatGPT? Accessed: 2023-08-18. [Online]. Available: https://www.bbc.com/news/business-65086798

[49] T. Claburn. (2023) ChatGPT creates mostly insecure code, but won't tell you unless you ask. Accessed: 2023-08-18. [Online]. Available: https://www.theregister.com/2023/04/21/chatgpt_insecure_code/

[50] L. Chen, M. Zaharia, and J. Zou, "How is ChatGPT's behavior changing over time?" *CoRR*, vol. abs/2307.09009, 2023.

[51] Y. Chen, A. E. Santosa, M. Y. Ang, A. Sharma, A. Sharma, and D. Lo, "A machine learning approach for vulnerability curation," in *MSR '20: 17th International Conference on Mining Software Repositories, Seoul, Republic of Korea, 29-30 June, 2020*, S. Kim, G. Gousios, S. Nadi, and J. Hejderup, Eds. ACM, 2020, pp. 32–42.

[52] P. McCorduck, *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*. A K Peters/CRC Press, 2004.

[53] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *20th IEEE/ACM International Conference on Automated Software Engineering (ASE 2005), November 7-11, 2005, Long Beach, CA, USA*, D. F. Redmiles, T. Ellman, and A. Zisman, Eds. ACM, 2005, pp. 273–282.

[54] Z. Zhang, Y. Lei, X. Mao, M. Yan, X. Xia, and D. Lo, "Context-aware neural fault localization," *IEEE Trans. Software Eng.*, vol. 49, no. 7, pp. 3939–3954, 2023.

[55] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in *Proceedings of the 20th International Symposium on Software Testing and Analysis, ISSTA 2011, Toronto, ON, Canada, July 17-21, 2011*, M. B. Dwyer and F. Tip, Eds. ACM, 2011, pp. 199–209.

[56] B. Johnson, Y. Song, E. R. Murphy-Hill, and R. W. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, D. Notkin, B. H. C. Cheng, and K. Pohl, Eds. IEEE Computer Society, 2013, pp. 672–681.

[57] M. Christakis and C. Bird, "What developers want and need from program analysis: An empirical study," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016*, D. Lo, S. Apel, and S. Khurshid, Eds. ACM, 2016, pp. 332–343.

[58] X. Xia, L. Bao, D. Lo, and S. Li, "'Automated debugging considered harmful' considered harmful: A user study revisiting the usefulness of spectra-based fault localization techniques with professionals using real bugs from large systems," in *2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016*. IEEE Computer Society, 2016, pp. 267–278.

[59] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. New York: Harper & Row, 1990.

[60] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann, "Software developers' perceptions of productivity," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, S. Cheung, A. Orso, and M. D. Storey, Eds. ACM, 2014, pp. 19–29.

[61] N. Forsgren, M. D. Storey, C. S. Maddila, T. Zimmermann, B. Houck, and J. L. Butler, "The SPACE of developer productivity: There's more to it than you think," *ACM Queue*, vol. 19, no. 1, pp. 20–48, 2021.

[62] G. Veletsianos, "Cognitive and affective benefits of an animated pedagogical agent: Considering contextual relevance and aesthetics," *Journal of Educational Computing Research*, vol. 36, no. 4, pp. 373–377, 2007.

[63] M. Anastassiu and G. Santos, "Resistance to change in software process improvement - an investigation of causes, effects and conducts," in *19th Brazilian Symposium on Software Quality, SBQS 2020, São Luís, Brazil, December, 2020*, D. Viana and M. Schots, Eds. ACM, 2020, p. 21.

[64] T. Amorim, A. Vogelsang, F. Pudlitz, P. Gersing, and J. Philipps, "Strategies and best practices for model-based systems engineering adoption in embedded systems industry," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, H. Sharp and M. Whalen, Eds. IEEE / ACM, 2019, pp. 203–212.

[65] P. R. Lawrence, "How to deal with resistance to change." *Harvard Business Review*, 1969.

[66] R. Minelli, A. Mocci, and M. Lanza, "I know what you did last summer: an investigation of how developers spend their time," in *Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, ICPC 2015, Florence/Firenze, Italy, May 16-24, 2015*, A. D. Lucia, C. Bird, and R. Oliveto, Eds. IEEE Computer Society, 2015, pp. 25–35.

[67] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Measuring program comprehension: A large-scale field study with professionals," *IEEE Trans. Software Eng.*, vol. 44, no. 10, pp. 951–976, 2018.

[68] N. Mangano, T. D. LaToza, M. Petre, and A. van der Hoek, "How software designers interact with sketches at the whiteboard," *IEEE Transactions on Software Engineering*, vol. 41, no. 2, pp. 135–156, 2014.

[69] H. H. Clark and S. E. Brennan, "Grounding in communication," in *Perspectives on socially shared cognition*, L. B. Resnick, J. M. Levine, and S. D. Teasley, Eds. American Psychological Association, 1991, pp. 127–149.

[70] A. D. Baddeley, "Working memory: looking back and looking forward," *Nature Reviews Neuroscience*, vol. 4, no. 10, pp. 829–839, 2003.

[71] E. Tulving and D. M. Thomson, "Encoding specificity and retrieval processes in episodic memory." *Psychological Review*, vol. 80, pp. 352–373, 1973.

[72] Y. Shi, Y. Yin, Z. Wang, D. Lo, T. Zhang, X. Xia, Y. Zhao, and B. Xu, "How to better utilize code graphs in semantic code search?" in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 722–733.

[73] X. Hu, G. Li, X. Xia, D. Lo, and Z. Jin, "Deep code comment generation," in *Proceedings of the 26th Conference on Program Comprehension, ICPC 2018, Gothenburg, Sweden, May 27-28, 2018*, F. Khomh, C. K. Roy, and J. Siegmund, Eds. ACM, 2018, pp. 200–210.

[74] A. Karpathy, "Software 2.0," Blog Post, November 2017. [Online]. Available: https://medium.com/@karpathy/software-2-0-a64152b37c35

[75] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Trans. Software Eng.*, vol. 33, no. 1, pp. 2–13, 2007.

[76] D. Lo, H. Cheng, J. Han, S. Khoo, and C. Sun, "Classification of software behaviors for failure detection: a discriminative pattern mining approach," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, J. F. E. IV, F. Fogelman-Soulié, P. A. Flach, and M. J. Zaki, Eds. ACM, 2009, pp. 557–566.

[77] M. Kruppa, "Google DeepMind CEO says some form of AGI possible in a few years," *The Wall Street Journal*, 2023, accessed: 2023-08-19. [Online]. Available: https://www.wsj.com/articles/google-deepmind-ceo-says-some-form-of-agi-possible-in-a-few-years-2705f452

[78] K. Grace, Z. Stein-Perlman, B. Weinstein-Raun, and J. Salvatier, "2022 expert survey on progress in AI," *AI Impacts*, Aug 2022, accessed: 2023-08-19. [Online]. Available: https://aiimpacts.org/2022-expert-survey-on-progress-in-ai/

[79] R. Parasuraman and V. Riley, "Humans and automation: Use, misuse, disuse, abuse," *Human Factors*, vol. 39, no. 2, pp. 230–253, 1997.

[80] J. D. Lee and K. A. See, "Trust in automation: Designing for appropriate reliance," *Human Factors*, vol. 46, no. 1, pp. 50–80, 2004.

[81] B. M. Muir, "Trust between humans and machines, and the design of decision aids," *Int. J. Man Mach. Stud.*, vol. 27, pp. 527–539, 1987.

[82] J.-Y. Jian, A. Bisantz, and C. Drury, "Foundations for an empirically determined scale of trust in automated systems," *International Journal of Cognitive Ergonomics*, vol. 4, pp. 53–71, 03 2000.

[83] K. A. Hoff and M. Bashir, "Trust in automation: Integrating empirical evidence on factors that influence trust," *Human Factors*, vol. 57, no. 3, pp. 407–434, 2015.

[84] Z. Wan, X. Xia, A. E. Hassan, D. Lo, J. Yin, and X. Yang, "Perceptions, expectations, and challenges in defect prediction," *IEEE Trans. Software Eng.*, vol. 46, no. 11, pp. 1241–1266, 2020.

[85] X. Hu, X. Xia, D. Lo, Z. Wan, Q. Chen, and T. Zimmermann, "Practitioners' expectations on automated code comment generation," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022.* ACM, 2022, pp. 1693–1705.

[86] C. Wang, J. Hu, C. Gao, Y. Jin, T. Xie, H. Huang, Z. Lei, and Y. Deng, "Practitioners' expectations on code completion," *CoRR*, vol. abs/2301.03846, 2023.

[87] B. Johnson, C. Bird, D. Ford, N. Forsgren, and T. Zimmermann, "Make your tools sparkle with trust: The PICSE framework for trust in software tools," in *45th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, SEIP@ICSE 2023, Melbourne, Australia, May 14-20, 2023.* IEEE, 2023, pp. 409–419.

[88] P. T. Devanbu, T. Zimmermann, and C. Bird, "Belief & evidence in empirical software engineering," in *Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016*, L. K. Dillon, W. Visser, and L. A. Williams, Eds. ACM, 2016, pp. 108–119.

[89] B. Xu, T. Nguyen, T. Le-Cong, T. Hoang, J. Liu, K. Kim, C. Gong, C. Niu, C. Wang, B. Le, and D. Lo, "Are we ready to embrace generative AI for software Q&A?" *CoRR*, vol. abs/2307.09765, 2023.

[90] J. Mökander, J. Schuett, H. R. Kirk, and L. Floridi, "Auditing large language models: a three-layered approach," *CoRR*, vol. abs/2302.08500, 2023.

[91] S. Park, S. Kim, and Y. Lim, "Fairness audit of machine learning models with confidential computing," in *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, F. Laforest, R. Troncy, E. Simperl, D. Agarwal, A. Gionis, I. Herman, and L. Médini, Eds. ACM, 2022, pp. 3488–3499.

[92] Q. Huang, X. Xia, and D. Lo, "Revisiting supervised and unsupervised models for effort-aware just-in-time defect prediction," *Empir. Softw. Eng.*, vol. 24, no. 5, pp. 2823–2862, 2019.

[93] T. Mende and R. Koschke, "Effort-aware defect prediction models," in *14th European Conference on Software Maintenance and Reengineering, CSMR 2010, 15-18 March 2010, Madrid, Spain*, R. Capilla, R. Ferenc, and J. C. Dueñas, Eds. IEEE Computer Society, 2010, pp. 107–116.

[94] X. Zhou, D. Han, and D. Lo, "Assessing generalizability of Code-BERT," in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2021, Luxembourg, September 27 - October 1, 2021.* IEEE, 2021, pp. 425–436.

[95] S. Lu, D. Guo, S. Ren, J. Huang, A. Svyatkovskiy, A. Blanco, C. B. Clement, D. Drain, D. Jiang, D. Tang, G. Li, L. Zhou, L. Shou, L. Zhou, M. Tufano, M. Gong, M. Zhou, N. Duan, N. Sundaresan, S. K. Deng, S. Fu, and S. Liu, "CodeXGLUE: A machine learning benchmark dataset for code understanding and generation," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, J. Vanschoren and S. Yeung, Eds., 2021.

[96] T. Zhang, I. C. Irsan, F. Thung, and D. Lo, "Cupid: Leveraging ChatGPT for more accurate duplicate bug report detection," *arXiv preprint arXiv:2308.10022*, 2023.

[97] A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, M. S. Aktar, and 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 2023, Oulu, Finland, June 14-16, 2023.* ACM, 2023, pp. 279–285.

[98] J. He, Z. Xin, B. Xu, T. Zhang, K. Kim, Z. Yang, F. Thung, I. Irsan, and D. Lo, "Representation learning for Stack Overflow posts: How far are we?" *arXiv preprint arXiv:2303.06853*, 2023.

[99] Z. Yang, J. Shi, J. He, and D. Lo, "Natural attack for pre-trained models of code," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022.* ACM, 2022, pp. 1482–1493.

[100] A. Mastropaolo, L. Pascarella, E. Guglielmi, M. Ciniselli, S. Scalabrino, R. Oliveto, and G. Bavota, "On the robustness of code generation techniques: An empirical study on GitHub Copilot," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023.* IEEE, 2023, pp. 2149–2160.

[101] M. Weyssow, X. Zhou, K. Kim, D. Lo, and H. A. Sahraoui, "On the usage of continual learning for out-of-distribution generalization in pre-trained language models of code," *CoRR*, vol. abs/2305.04106, 2023.

[102] X. Zhou, K. Kim, B. Xu, J. Liu, D. Han, and D. Lo, "The devil is in the tails: How long-tailed code distributions impact large language models," in *IEEE/ACM International Conference on Automated Software Engineering, ASE 2023.* IEEE, 2023.

[103] 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. B. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "Graph-CodeBERT: Pre-training code representations with data flow," in *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021.* OpenReview.net, 2021.

[104] Y. Gui, Y. Wan, H. Zhang, H. Huang, Y. Sui, G. Xu, Z. Shao, and H. Jin, "Cross-language binary-source code matching with intermediate representations," in *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022, Honolulu, HI, USA, March 15-18, 2022.* IEEE, 2022, pp. 601–612.

[105] C. Niu, C. Li, V. Ng, D. Lo, and B. Luo, "Fair: Flow type-aware pre-training of compiler intermediate representations," in *IEEE/ACM International Conference on Software Engineering, ICSE 2024.* ACM, 2024.

[106] T. Ahmed and P. T. Devanbu, "Few-shot training LLMs for project-specific code-summarization," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022.* ACM, 2022, pp. 177:1–177:5.

[107] T. Ahmed, K. S. Pai, P. Devanbu, and E. T. Barr, "Improving few-shot prompts with relevant static analysis products," *arXiv preprint arXiv:2304.06815*, 2023.

[108] H. Joshi, J. P. C. Sánchez, S. Gulwani, V. Le, G. Verbruggen, and I. Radicek, "Repair is nearly generation: Multilingual program repair with LLMs," in *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, B. Williams, Y. Chen, and J. Neville, Eds. AAAI Press, 2023, pp. 5131–5140.

[109] Z. Fan, X. Gao, M. Mirchev, A. Roychoudhury, and S. H. Tan, "Automated repair of programs from large language models," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023.* IEEE, 2023, pp. 1469–1481.

[110] Y. Peng, C. Wang, W. Wang, C. Gao, and M. R. Lyu, "Generative type inference for Python," *CoRR*, vol. abs/2307.09163, 2023.

[111] B. Rozière, J. Zhang, F. Charton, M. Harman, G. Synnaeve, and G. Lample, "Leveraging automated unit tests for unsupervised code translation," in *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022.* OpenReview.net, 2022.

[112] C. S. Xia and L. Zhang, "Keep the conversation going: Fixing 162 out of 337 bugs for $0.42 each using ChatGPT," *CoRR*, vol. abs/2304.00385, 2023.

[113] Y. Liu, J. Lin, O. Anuyah, R. A. Metoyer, and J. Cleland-Huang, "Generating and visualizing trace link explanations," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022.* ACM, 2022, pp. 1033–1044.

[114] L. Li, S. H. H. Ding, Y. Tian, B. C. M. Fung, P. Charland, W. Ou, L. Song, and C. Chen, "VulANalyzeR: Explainable binary vulnerability detection with multi-task learning and attentional graph convolution," *ACM Trans. Priv. Secur.*, vol. 26, no. 3, April 2023.

[115] P. Mahbub, O. Shuvo, and M. M. Rahman, "Explaining software bugs leveraging code structures in neural machine translation," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*. IEEE, 2023, pp. 640–652.

[116] R. Widyasari, G. A. A. Prana, S. A. Haryono, Y. Tian, H. N. Zachiary, and D. Lo, "XAI4FL: Enhancing spectrum-based fault localization with explainable artificial intelligence," in *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension, ICPC 2022, Virtual Event, May 16-17, 2022*, A. Rastogi, R. Tufano, G. Bavota, V. Arnaoudova, and S. Haiduc, Eds. ACM, 2022, pp. 499–510.

[117] A. H. Mohammadkhani, N. S. Bommi, M. Daboussi, O. Sabnis, C. Tantithamthavorn, and H. Hemmati, "A systematic literature review of explainable AI for software engineering," *CoRR*, vol. abs/2302.06065, 2023.

[118] T. Avgerinos, S. K. Cha, A. Rebert, E. J. Schwartz, M. Woo, and D. Brumley, "Automatic exploit generation," *Commun. ACM*, vol. 57, no. 2, pp. 74–84, 2014.

[119] E. Iannone, D. D. Nucci, A. Sabetta, and A. D. Lucia, "Toward automated exploit generation for known vulnerabilities in open-source libraries," in *29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, Madrid, Spain, May 20-21, 2021*. IEEE, 2021, pp. 396–400.

[120] H. J. Kang, T. G. Nguyen, B. Le, C. S. Pasareanu, and D. Lo, "Test mimicry to assess the exploitability of library vulnerabilities," in *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022*, S. Ryu and Y. Smaragdakis, Eds. ACM, 2022, pp. 276–288.

[121] F. Thung, D. Lo, and J. Lawall, "Automated library recommendation," in *20th Working Conference on Reverse Engineering, WCRE 2013, Koblenz, Germany, October 14-17, 2013*, R. Lämmel, R. Oliveto, and R. Robbes, Eds. IEEE Computer Society, 2013, pp. 182–191.

[122] S. Chattopadhyay, L. K. Chong, A. Roychoudhury, T. Kelter, P. Marwedel, and H. Falk, "A unified WCET analysis framework for multicore platforms," *ACM Trans. Embed. Comput. Syst.*, vol. 13, no. 4s, pp. 124:1–124:29, 2014.

[123] A. Budi, D. Lo, L. Jiang, and Lucia, "*kb*-anonymity: a model for anonymized behaviour-preserving test and debugging data," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, M. W. Hall and D. A. Padua, Eds. ACM, 2011, pp. 447–457.

[124] Lucia, D. Lo, L. Jiang, and A. Budi, "kbe-anonymity: test data anonymization for evolving programs," in *IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012*, M. Goedicke, T. Menzies, and M. Saeki, Eds. ACM, 2012, pp. 262–265.

[125] F. Peters, T. Menzies, L. Gong, and H. Zhang, "Balancing privacy and utility in cross-company defect prediction," *IEEE Trans. Software Eng.*, vol. 39, no. 8, pp. 1054–1068, 2013.

[126] Z. Yang, Z. Zhao, C. Wang, J. Shi, D. Kim, D. Han, and D. Lo, "What do code models memorize? an empirical study on large language models of code," *CoRR*, vol. abs/2308.09932, 2023.

[127] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 2017, pp. 3–18.

[128] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1*. IEEE, 2015, pp. 57–64.

[129] J. Shi, Z. Yang, B. Xu, H. J. Kang, and D. Lo, "Compressing pretrained models of code into 3 MB," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 2022, pp. 24:1–24:12.

[130] S. Shanbhag and S. Chimalakonda, "Exploring the under-explored terrain of non-open source data for software engineering through the lens of federated learning," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 1610–1614.

[131] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. IEEE Computer Society, 2015, pp. 463–480.

[132] C. Metz. (2022) Lawsuit takes aim at the way A.I. is built. Accessed: 2023-08-19. [Online]. Available: https://www.nytimes.com/2022/11/23/technology/copilot-microsoft-ai-lawsuit.html

[133] M. Ballhausen, "Free and open source software licenses explained," *Computer*, vol. 52, no. 6, pp. 82–86, 2019.

[134] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Trans. Software Eng.*, vol. 28, no. 7, pp. 654–670, 2002.

[135] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, "DECKARD: scalable and accurate tree-based detection of code clones," in *29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007*. IEEE Computer Society, 2007, pp. 96–105.

[136] V. Saini, F. Farmahinifarahani, Y. Lu, P. Baldi, and C. V. Lopes, "Oreo: detection of clones in the twilight zone," in *Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018*, G. T. Leavens, A. Garcia, and C. S. Pasareanu, Eds. ACM, 2018, pp. 354–365.

[137] N. Mehrotra, N. Agarwal, P. Gupta, S. Anand, D. Lo, and R. Purandare, "Modeling functional similarity in source code with graph-based siamese networks," *IEEE Trans. Software Eng.*, vol. 48, no. 10, pp. 3771–3789, 2022.

[138] F. Gao, Y. Wang, and K. Wang, "Discrete adversarial attack to models of code," *Proc. ACM Program. Lang.*, vol. 7, no. PLDI, pp. 172–195, 2023.

[139] A. Jha and C. K. Reddy, "CodeAttack: Code-based adversarial attacks for pre-trained programming language models," in *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, B. Williams, Y. Chen, and J. Neville, Eds. AAAI Press, 2023, pp. 14 892–14 900.

[140] Z. Yang, B. Xu, J. M. Zhang, H. J. Kang, J. Shi, J. He, and D. Lo, "Stealthy backdoor attack for code models," *CoRR*, vol. abs/2301.02496, 2023.

[141] T.-D. Nguyen, Y. Zhou, X. B. D. Le, Patanamon, Thongtanunam, and D. Lo, "Adversarial attacks on code models with discriminative graph patterns," *CoRR*, vol. abs/2308.11161, 2023.

[142] Y. Wan, S. Zhang, H. Zhang, Y. Sui, G. Xu, D. Yao, H. Jin, and L. Sun, "You see what I want you to see: poisoning vulnerabilities in neural code search," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 1233–1245.

[143] R. Schuster, C. Song, E. Tromer, and V. Shmatikov, "You autocomplete me: Poisoning vulnerabilities in neural code completion," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 1559–1575.

[144] D. Jarman, J. Berry, R. Smith, F. Thung, and D. Lo, "Legion: Massively composing rankers for improved bug localization at adobe," *IEEE Trans. Software Eng.*, vol. 48, no. 8, pp. 3010–3024, 2022.

[145] X. D. Le, T. B. Le, and D. Lo, "Should fixing these failures be delegated to automated program repair?" in *26th IEEE International Symposium on Software Reliability Engineering, ISSRE 2015, Gaithersbury, MD, USA, November 2-5, 2015*. IEEE Computer Society, 2015, pp. 427–437.

[146] T. B. Le, F. Thung, and D. Lo, "Will this localization tool be effective for this bug? mitigating the impact of unreliability of information retrieval based bug localization tools," *Empir. Softw. Eng.*, vol. 22, no. 4, pp. 2237–2279, 2017.

[147] M. Golagha, A. Pretschner, and L. C. Briand, "Can we predict the quality of spectrum-based fault localization?" in *13th IEEE International Conference on Software Testing, Validation and Verification,*

ICST 2020, Porto, Portugal, October 24-28, 2020. IEEE, 2020, pp. 4–15.

[148] V. Satir, J. Banmen, J. Gerber, and M. Gomori, *The Satir Model: Family Therapy & Beyond.* Science & Behavior Books, 1991.

[149] L. Lindstrom and K. Beck, "It gets worse before it gets better: Changing to XP," *Cutter IT Journal*, vol. 16, no. 2, pp. 12–17, 2003.

[150] Y. Tian, D. Lo, and C. Sun, "Information retrieval based nearest neighbor classification for fine-grained bug severity prediction," in *19th Working Conference on Reverse Engineering, WCRE 2012, Kingston, ON, Canada, October 15-18, 2012.* IEEE Computer Society, 2012,

pp. 215–224.

[151] C. Niu, C. Li, V. Ng, and B. Luo, "CrossCodeBench: Benchmarking cross-task generalization of source code models," in *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023.* IEEE, 2023, pp. 537–549.

[152] D. Ko, J. Choi, H. K. Choi, K. On, B. Roh, and H. J. Kim, "MELTR: meta loss transformer for learning to fine-tune video foundation models," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2023, Vancouver, BC, Canada, June 17-24, 2023.* IEEE, 2023, pp. 20 105–20 115.