# Software Engineering by and for Humans in an AI Era

SILVIA ABRAHÃO, JOHN GRUNDY, MAURO PEZZÈ, MARGARET-ANNE STOREY, and DAMIAN ANDREW TAMBURRI

The landscape of software engineering is undergoing a transformative shift driven by advancements in machine learning, artificial intelligence (AI), and autonomous systems. This roadmap paper explores how these technologies are reshaping the field, positioning humans not only as end users but also as critical components within expansive software ecosystems. We examine the challenges and opportunities arising from this human-centered paradigm, including ethical considerations, fairness, and the intricate interplay between technical and human factors. By recognizing humans at the heart of the software lifecycle —spanning professional engineers, end users, and end-user developers —we emphasize the importance of inclusivity, human-aligned workflows, and the seamless integration of AI-augmented socio-technical systems. As software systems evolve to become more intelligent and human-centric, software engineering practices must adapt to this new reality. This paper provides a comprehensive examination of this transformation, outlining current trends, key challenges, and opportunities that define the emerging research and practice landscape, and envisioning a future where software engineering and AI work synergistically to place humans at the core of the ecosystem.

**ACM Reference Format:**
Silvia Abrahão, John Grundy, Mauro Pezzè, Margaret-Anne Storey, and Damian Andrew Tamburri. 2015. Software Engineering by and for Humans in an AI Era. *ACM Trans. Softw. Eng. Methodol.* 52, 4, Article 111 (August 2015), 45 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 INTRODUCTION

The last decade has seen a sudden acceleration of many new technologies that radically change software production and its impact on human life, work and society. Generative AI, extended reality and the evolving internet of things have opened major new frontiers. These have upset current and future research and practice with a disruptive revolution greatly beyond the significant changes of even the last decade. Generative AI has already redefined the production of software systems with improvement of productivity and sometimes disruptive side effects [76, 85]. Current tools and practices are just scratching the surface of a deep revolution that may have major impacts on the way we engineer software systems in the next decades [46, 58, 128]. Generative AI, extended reality and the internet of things enable new ways for humans to interact with their diverse software systems. Machine-learning-powered systems and bots autonomously evolve and interact with humans in many ways that were unforeseen till few years ago. Increasingly humans are begining to interact more seamlessly and unconsciously with a range of complex software systems simply by living in increasingly software-driven smart cities. Extended reality and the internet of things enable human-system interactions far beyond the classic keyboard-mouse-display interactions.

Generative AI empowers new scenarios for both designing software systems and living within human-centric systems. However, these developments are not without a variety of risks, including concerns over what SE jobs will

Authors' address: Silvia Abrahão; John Grundy; Mauro Pezzè; Margaret-Anne Storey; Damian Andrew Tamburri.


Manuscript submitted to ACM

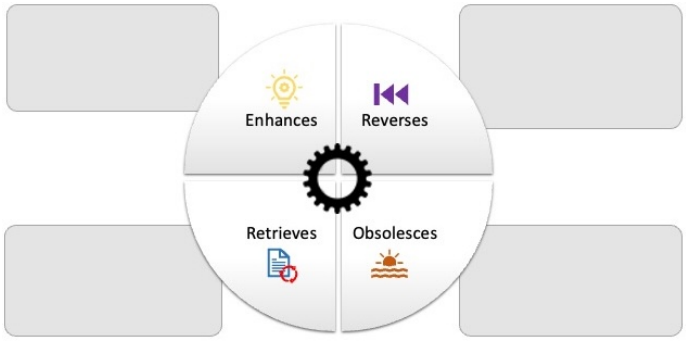



Fig. 1. Mc Luhan's Tetrad

look like, AI bias, ethical and responsible engineering practices, inclusive solutions and how to best educate future AI-powered system engineers [86, 92, 130, 147].

In the *Artificial Intelligence for Software Engineering: The Journey so far and the Road ahead* paper in this issue, Chatzigeorgiou et al. [28] discuss the impact of AI on software engineering practices. In this paper we focus on human factors in the *area of AI*, and propose a roadmap for research on human aspects in both SE development processes and human-centric software systems. We discuss the way new AI-powered tools dramatically change the development of software systems and the dynamics of development teams, the SE profession and SE education. We discuss the need for software adaptation for diverse end users and usage contexts by leveraging AI and human interaction, and the validation of human-centric self-adaptive ecosystems. We argue how human-centric, self-adaptive ecosystems reshape the role of software in human life and society.

We illustrate the impact of Generative AI and human-centric self-adaptive ecosystems throughout Sections 2 and 3, with McLuhan's Tetrad [110]. The *Tetrad* is a diagram that the Canadian philosopher Herbert Marshall McLuhan defined in the mid seventies to explain the disruptive effect of media on the society. McLuhan observed that disruptive technologies always follow intrinsic laws: they amplify certain human abilities, render previous technologies obsolete, and when pushed to their extremes, create a need for new innovations to address emergent new challenges. In this paper we use McLuhan's Tetrad to highlight some of the key effects of disruptive innovation, as Storey et al. suggest in their *disruptive research playbook* [152]. Figure 1 shows the four diamonds that illustrate the effect of the technology (in the center in the diagram) by indicating how the new technology *enhances* software development and systems, *obsolesces* technologies and approaches, *reverses* common practice, and *retrieves* past results, to propose a roadmap for software engineering by and for humans systems in the AI era. For example, the Internet, in the middle of Figure 1, has amplified distributed development (the enhances diamond), and has driven out of practice tarballs and zip files (the obsolesces diamond), by offering a new support to collaborative development, it has recovered standardized processes to organize distributed development (the retrieves diamond) and it has emphasized social coding when pushed to the limit (the reverses diamond).

## 2 SOFTWARE DEVELOPMENT IN THE ERA OF AI

Software development is one of the most creative, complex but also rewarding forms of knowledge work. Facilitated by the internet, tens of thousands of developers are able to work together, directly and indirectly, to produce long-lived

multi-version complex systems (such as Windows and Google) with millions of static and dynamic system and software dependencies that change rapidly over time to keep up with the demand for new or different user features.

To keep pace with such complexity, developers innovate and adopt new processes, novel technologies and powerful automation to augment their capabilities for creating and orchestrating how software system versions are managed by these distributed networks of thousands of developers. These innovations, such as the internet and social coding tools, and processes, such as agile, change not just the speed and quality of the software being developed, but they also alter the very essence of how software is authored by developers, and the ways they work together.

Generative AI-based tools are the latest disruption to reshape how software is being engineered. We use McLuhan's Tetrad, introduced in Section 1, as a lens for laying out a multi-path roadmap of how AI will disrupt development practices and experiences in expected and unexpected ways, as well as potentially both positive and negative ways (see Figure 2). We may anticipate that AI may **enhance** the speed and creativity of developers, and remove (makes **obsolete**) friction and tedium in work and the need to know lower level technical details. However, it may also **retrieve** artifacts and activities from times before - for instance, interaction with ICQ style chatbots and more reliance on "natural language" as the design language. What it will **reverse** into is also of concern - loss of control, trust, and quality over time.

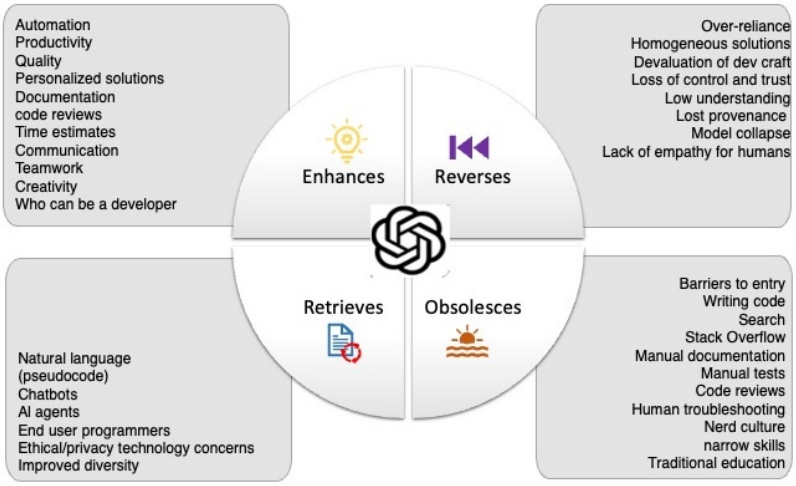



Fig. 2. An idealised view of the disruptive effects of AI

In the following we first lay out how AI is reshaping how we conceptualize developer productivity and experience (Section 2.1), next we explore how AI is disrupting the tools used in software engineering (Section 2.2), how AI is transforming how humans and machines together create software collaboratively (Section 2.3), how AI will impact *who* will be designing software and the software engineering profession (Section 2.4) and finally how AI will impact software engineering education (Section 2.5).

### 2.1 The Impact of AI on the Developer Productivity and Experience

Improving tools and processes to enhance developer productivity and their experience, has been an ongoing quest since even before the term "software engineering" was coined in the NATO 1968 conference on the software crisis [120].

Before we discuss how AI is impacting developer tooling (see Section 2.2), we revisit some historical landmarks in developer productivity and experience, the ongoing challenges we face today and a roadmap for future work to be considered about how AI impacts developer productivity and experience.

*2.1.1 Trends and Historical Landmarks.* Frederick J. Brooks was one of the first authors to write about developer productivity in his famous 1975 *Mythical Man Month* book [23][1]. The essays in his book, based on his experience managing a large, complex project at IBM on the OS/360, led to several timeless project management insights that still hold today. He recognized that the inherent essential complexity in software cannot be tamed by simply adding more developers to the project, in part due to communication and coordination overhead and difficulties maintaining an understanding of the overall design of a complex project. Brooks also famously noted that there is likely no *silver bullet* to significantly improve developer productivity. He argues that tools, languages and processes can only help us address what he termed accidental complexity in software, but do little to address the essential complexity inherent in the problem and solution domains. Brooks also discussed how software is not visible until there is an early prototype, and it is only with these early prototypes, that users will discover what they want, putting pressure on the requirements and the software to change. Brooks recognized many of the challenges we still experience with versioning, documentation, project estimation, administrative overhead and increasing errors over time.

Gerald Weinberg, another luminary, also discussed some of the same important themes about human factors in his landmark 1971 *The Psychology of Programming* book [170], and noted that software development is a human activity, rather than a technical activity. He talks about the psychological challenges in understanding program complexity, and the importance of effective communication and positive team dynamics for distinguishing high performing software teams. He also mentioned the importance of continuous improvement, made possible by learning during debugging. He further proposed innovative ideas for developer documentation, whereby developers would write customized answers to specific developer questions that capture the context and needs of the questioning developers. This idea is prescient of Stack Overflow, a tool that did not appear for almost 40 years. The concepts behind continuous software engineering and collaborative and interactive learning are still the cornerstones of successful development today.

The argument that software development is much more than the technical details captured by code, was also strongly advocated by Peter Naur in his famous 1986 essay [119]. Naur considered software as *theories in the minds of developers*: theories about the problem domain, theories about how the software implements solutions to those problems, and theories about how the program can be evolved to support new and changing features. He also recognized the importance of mentorship and knowledge sharing among team members.

Another seminal book on software developer productivity was DeMarco and Lister's 1987 *Peopleware: Productive Projects and Teams* book [36]. DeMarco and Lister discussed the importance of developer psychology, team work and the developer's environment. They also recognized the importance of project management and the critical role of the manager, and how intrinsic motivation, driven by a developer's personal interest and satisfaction in their work, is an important key to a team's success (even more so than bonuses and extrinsic rewards).

During the 1990's and into the 2000's and beyond, practitioners and researchers, building on the recognition of the importance of human factors and teamwork, synthesized and shared wisdom to improve programming techniques [79], more advanced programming languages [87] and programming tools (see Section 2.2). Extreme programming [17] and the agile movement [148] further led to insights of how to improve developer productivity and their experience, with practices such as refactoring, pair programming, customer focus, communication, test driven development, testing,

[1]This is the reference to the first of several editions

scrum and continuous integration showing widespread adoption with positive impacts. While at the same time, others paid significant attention to how to measure, and hopefully, better control or influence productivity [15, 83, 155].

In the past decade, there has been additional substantive research on developer productivity [164], developer happiness [70] and satisfaction [154], developer motivation [18] and developer experience [72], and widespread adoption in industry of approaches for framing and measuring productivity [51, 52].

*2.1.2 Open Challenges.* AI dramatically impacts on developer productivity, experience, flow and creativity.

*Productivity:* How to define, understand and measure developer productivity is an ongoing challenge, tackled by both industry and research. Not that long ago, many large companies used lines of code as a way of measuring development productivity. We have come a long way from then and realized that productivity is a complex concept and if it is to be measured (some argue it should not be!), it should be measured across many dimensions. Forsgren, Storey and collaborators [52] proposed that productivity should be conceptualized across at least five dimensions: developer satisfaction and well-being, performance and quality, activity (such as lines of code, code reviews done), collaboration and communication, and finally developer/project efficiency and flow. The SPACE framework emerged from decades of research on developer productivity, and it has been adopted across the industry to understand different dimensions behind improvement interventions as well as ways of measurement. SPACE has also been used as a framework for understanding the different impacts as AI (implemented with Copilot) was adopted by GitHub developers [85]. But much remains to understand and improve how AI impacts developer productivity and developer experience. The dimensions in SPACE are one way to capture those insights, but as AI is adopted, we will expect to see new ways to measure these dimensions of productivity.

*Trust and Autonomy:* Important aspects of developer experience will relate to their trust in generative AI (and how it will impact the quality of their work) as well as their autonomy over how any automated developed work is done. Trust will also be influenced by how developers feel secure in how they use AI to generate code and other artifacts, as well as how using AI will influence their sense of identity [77].

*Creativity:* Software development is a highly creative endeavour. Researchers in our community are just starting to question how AI will enhance (or impede) creativity in the software activities of brainstorming, design, development and analysis [81]. When developers work with AI, will they become less or more creative? How can AI support creativity at the individual and team levels, while also helping developers, stay on track with their assigned work. Jackson *et al.* speculate that individual creativity can be enhanced by bringing in inspirations from other disciplines, bringing perspectives from the future users of the software, and relating those ideas to architecture design issues [81]. These authors also suggest that Generative AI may level the playing field and make individual personality traits less relevant, but at the same time may reduce reliance on human mentorship. How AI may further engage discussions between the end users and developers is discussed further below in this paper when we consider how generative AI will impact collaboration (see Section 2.3) and is an important open challenge.

*2.1.3 Roadmap.* There are many open research issues about developers' productivity and experience, and the industry is hungry to know how to conceptualize these concepts and how to measure the impact of tools, processes and practices. Specifically, the challenges mentioned above lead to the following research activities.

- **Which theories should we borrow and extend from other disciplines, and what new theories do we need to develop?** Developer productivity and developer experience both rely on concepts from many other

disciplines, including the management sciences, HCI, behavioural psychology and the social sciences. At the same time, we will have to build brand new theories, and revisit older theories as AI disrupts what we know and assume about developer productivity and their experience. For example, how AI will impact developer creativity is a topic of interest[2]. We will need to consult and be aware of theories of creativity from other disciplines as suggested by Jackson *et al.* [81].

- **How to measure developer productivity and experience?** As AI is rolled out, and as it extends its reach and capabilities, we need to know how to measure its impact on developer productivity, across many different dimensions such as those captured by the SPACE framework and on developer experience. We likely need new metrics and new ways of measuring and presenting impacts on productivity, experience, flow, creativity, trust and autonomy on developers, managers and researchers.
- **How will AI-powered tools and developer agents impact the creative side of software development processes and work?** Creativity is an often overlooked aspect of SE work. We need to study how AI-powered developer agents impact this, especially in terms of human collaboration around producing creative solutions to currently unknown, unsolved software solutions, domains, users and usage contexts. Will they dampen or improve creativity, and how can we ensure the latter vs the former.

### 2.2 The Impact of AI on Software Development Tools

*2.2.1 Software Development Trends Over Time.* Software developers are always on the forefront of either developing or adopting new technical innovations to support and transform their work. Indeed, software engineering has been assisted by automation since the first assembly language tools were constructed.

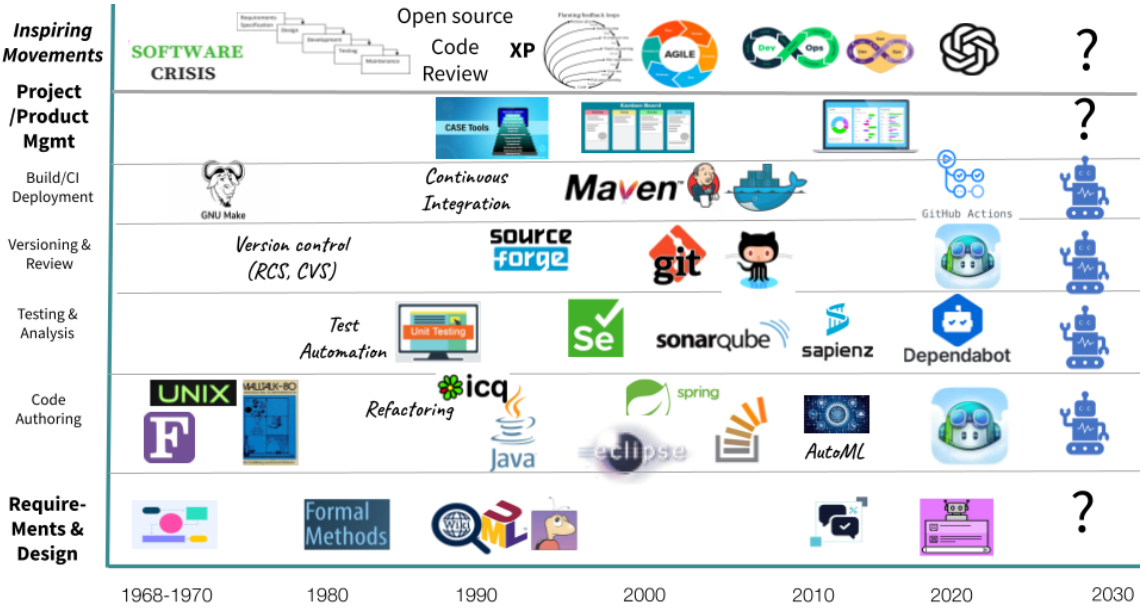
There are many different innovations that have emerged over time that support the inner loop of how developers write and test code, as well as innovations that support the outer loop of going from idea to deployed feature. Figure 3 shows a historical timeline of emerging tools that support the different phases of the software development lifecycle (both the inner and outer loops). Tools that support requirements elicitation, their capture and formalization; tools and languages that support and enhance how code is authored or auto-generated; tools that help ensure the quality of current and future work through testing and analysis; tools that allow developers to review each other's work and work together by versioning their work and systems; tools that automate the build and deployment processes; and tools that help manage the project management process.

The figure clusters tools according to their main supportive role. All tools followed an arc that was shaped by inspiring movements in how software engineering should be done (see the top row of Figure 3). For example, the recognition that there was a looming software crisis in the late 60's spurred the need for tools to manage software complexity and support the then recognized inevitable need for software to continually evolve. The agile movement recognized the importance of humans and how they communicate and co-create in the process of software development led to tools that emphasized social interactions and distributed development. The Devops movement, as another example, influenced the development of continuous integration pipelines and tooling.

Over the past several years, the latest disruptive movement of generative AI has led to an increasing number of AI-powered software engineering support tools. The suggestion to use AI in developer tools is also not new. Rich *et al.* back in 1988 proposed the "programmer's apprentice", a vision of how tooling could use AI to help developers complete a wide range of tasks [136]. Although much of this early vision was not fully realized, of late, AI is mow leaving its

[2]This topic will be explored in an upcoming Dagstuhl seminar, see https://www.dagstuhl.de/25412

mark on code authoring, testing, code review, build, deployment, security and documentation tools among others. AI is reshaping the entire tooling environment in which software engineers practice. The right side of Figure 3 indicates that AI and the use of agents or automation is likely to transform all tools across the development life cycle, with new tools that we cannot even conceive of yet likely emerging.




The needs driven emergence of innovations in software developer tools that are shaped by important movements in software engineering (such as agile, devops and recently AI), with rows below representing the outer loop (requirements/design and project management) and inner loop (from code to deployment). This figure also shows the uncertainty of how these tools will evolve by 2030. For the inner loop, we expect that automation will play a bigger role by 2030, but expect humans to continue playing a large role for outer loop but be assisted by AI and collaborate with AI agents.

Fig. 3. Historic timeline of tools supporting the software engineering lifecycle

*2.2.2 Open Challenges.* As software engineering systems are augmented with AI capabilities, the new opportunities they bring also lead to several open challenges. We describe these opportunities and challenges using the different categories of tools used across the entire life cycle of software development (as displayed in Figure 3).

*Requirements Elicitation, Management and Documentation:* The mechanisms for requirements elicitation, management and documentation are predicted to go through several transformative and disruptive changes due to the adoption of generative AI and Natural Language Processing [3]. Formerly, eliciting and documenting requirements was a labour intensive task involving time consuming customer inquiries, and the negotiation of competing requests and design activities. Validating and modeling requirements (to ensure completeness and consistency), and tracing of requirements to code and other artifacts were also time consuming and error prone tasks. Indeed, more often than not, requirements were never written down, and rarely explicitly linked to associated code artifacts.

[3]https://medium.com/@workboxtech/ai-in-requirements-engineering-clarity-and-consistency-boost-1c7fbf2d28dd

The industry has started using generative AI to model user personas and to check for consistency and compliance among requirements, conversational bots to assist in eliciting requirements, analysis of text in feedback forums and social media to identify new and changing features, text generation to assist in documentation, links to show the traceability of requirements to code, rapid prototyping to assist in brainstorming of new requirements, and the use of generative AI to help in predicting requirements to improve non functional requirements such as security, privacy and performance. These new tool features are just some of the possibilities that are just around the corner.

In the academic domain, the latest conference on requirements engineering showcases many papers on how AI can be used across the field of requirements engineering[4], while the literature review by Hou et al. mentions several papers that have explored how LLMs are used in requirements engineering to date (key topics include: requirements disambiguation inherent in natural language, coherence across stakeholders, requirements classification, and traceability automation) [78].

These new capabilities and opportunities, although exciting, bring new challenges - some of which may be hard to predict. More rapid elicitation of requirements puts more pressure on the design and delivery of code to meet those needs, in turn putting pressure on quality. AI assisted or automated modeling of users and requirements, automated documentation and validation may lead to a lack of human control and oversight or recognition of biases that may emerge from the training data used. Verification of the suggested requirements and their documentation will need to be made by the "humans" involved. Designing with diversity in mind requires special attention that may be missed when large language models are used. Furthermore, although AI could be used to help address security concerns, using AI in the requirements elicitation process may introduce new threat attack vectors that may be close to impossible for the humans involved to detect without proactive monitoring.

*Coding Velocity and Ease:* There are already well established tools (for instance, Copilot from GitHub) and results from empirical studies to indicate that the use of generative AI for code generation helps improve the speed of coding [85] and there are well over 100 published academic papers on the topic of code generation alone [78].

Generative AI techniques can assist in generating snippets of code, removing the need for developers to write tedious blocks of code or the need to know how to use frameworks or APIs, and also has shown potential in generating code not just from text prompts in chat interfaces but also from sketches of user interfaces, or from other examples of user interactions (for instance, from user stories). Beyond code generation, many techniques also assist in program synthesis (generating code from formal requirements), code completion, code documentation, API's, code summarizations and search. There has also been great strides in the area of program repair (generating patches for bugs in maintenance tasks), especially when generative AI is combined with static analysis [175].

The increases in coding velocity and ease for both new, evolving and legacy code [78], but may introduce challenges in terms of software quality, feature and code bloat, security risks and unnecessary technical debt. Although generative AI may increase velocity, over time it may slow things down even more[5] as well as have disastrous effects on the environment if care is not taken in how large language models and other expensive AI techniques are used. A bigger concern may be the eventual need for a "developer" to remain in the loop when things go wrong or when the automation or AI breaks down. In these cases, having techniques to help a human debug, understand and repair the AI generated or repaired code, may be essential given the scale and complexity of new code that is likely to be generated. That is, tools to help address the new cognitive challenges AI SE coding tools may introduce is essential to consider up front.

[4]https://conf.researchr.org/home/RE-2024

[5]just as the introduction of the car sped up transportation until too many cars were on the road

Finally, the focus to date has predominantly been on development speed, but there is a need to study more on the impact of these tools on developer flow experiences and their motivation [94], as well as how these tools impact developer learning of new skills [77].

*Testing and Debugging:* How software is tested and analyzed is undergoing rapid changes also due to the use of generative AI. Tests and test data can be generated from natural language requirements, from formal specifications, from prompts, from conversations with bots, from code, or from similar systems, and from user interactions. These tests, unit, integration, and end-to-end tests, can be used to safeguard against regressions as future changes are made, testing both functional and non-functional requirements such as security. Indeed, large language models are already being used for vulnerability detection [78] and compliance violations during development tasks.

Generative AI can be used to determine ideal test configurations and to take into consideration performance issues when building software, and to suggest architectural changes to the underlying system to facilitate more efficient testing and analysis over time. Generative AI shows great potential in bug localization [97].

But there are some risks and challenges to be considered across the use of AI in testing, analysis and debugging activities. There may, over time, be too much reliance and trust on generated tests and bug detection/localizations, and the data that the generated tests and recommendations are trained from may be limited, especially in the face of a rapidly evolving software system and user base. Developer knowledge about how to test and debug may erode over time, and they may need new skills to use the AI to effectively test and debug tomorrow's software systems. Debugging, in the face of subtle errors, may be substantially more difficult when there is a complex and not well understood testing or verification framework being used. Safeguarding also against the "bystander" effect is something to be wary about. Developers may become complacent over time and possibly ignore signals that indicate things are not working right or that there have been security concerns introduced during the process of testing, quality analysis and debugging. Understanding human aspects of this new generation of testing and debugging tools is critical.

*Continuous Integration and Deployment:* Modern software development relies on continuous integration to support hundreds (and even thousands) of developers pushing changes to multiple versions of software across one or more software repositories. Ensuring that developers can work on the system at the same time (or on parts others depend on), is made possibly by build tools and merge conflict features. Tools such as GitHub further support developers collaborating and sharing insights on what they are working on and the changes that have been done, and the rationale for these changes, through work items and issues. Continuous integration is a socio-technical activity [43] that requires the orchestration of both human and tools to achieve speed and quality advantages it provides.

Code review is an essential practice to ensure that developers check and validate each others' changes. Code review is not just about verifying code, but it is also about sharing awareness and knowledge of what others are doing and knowledge about some of the code as it is developed over time. No single developer will know the entire system, but having multiple developers work on and review the same parts of the code, means that knowledge is distributed in a way that supports change over time. However, code review is often a bottleneck in rapid software development, and much code review work is to check for trivial matters. Because of this, there is some ongoing research on how AI can automate or augment code review activities. However, the limitation of using LLMs to automate or augment code review, is that it may reduce the other human and social benefits attained from doing manual code review - the advantages of knowledge transfer, skill acquisition, awareness of others' work, connections made between developers, and co-ownership across a code base may be reduced.

*Project and Product Management:* The use of generative AI for project and product management is an emerging topic of interest across the industry with limited attention by researchers so far [78]. Generative AI shows potential to support planning, cost estimations, tracking of work, communication, coordination, and decision making [78]. Some of the challenges for these tools are similar to challenges we already mentioned above including over reliance on automation, lack of accountability, bias and reduced oversight by humans and lack of communication and awareness. When any system is automated, end users often suffer at the expense of more efficiency for the producers or owners of the system. The reduction in human communication ability to deal with "exceptions" often leads to disastrous friction for the end users. These concerns need to be considered if and when AI is used for automating parts of the project management process.

*2.2.3 Roadmap.* Extensive research efforts are already well underway to study the impact of AI on the tooling landscape that supports the entire software engineering life-cycle. Inspired by McLuhan's laws of technology that are captured by his tetrad, we frame some future research avenues our community can continue or start to explore in terms of the different ways AI can be used to support software engineering activities and how we can go about understanding its impact.

- **How to Leverage Software Knowledge to Improve LLMs for SE?** Recent research on how to go beyond treating programs as text [175], and that leverage program presentation techniques, dynamic analysis and semantic analysis show great potential for improving how LLMs will support code generation, program synthesis, program repair, program comprehension, software evolution, code review, feature testing, security testing and build performance. That is, these techniques, perhaps put aside as AI was first introduced, should be "brought back" (to use McLuhan's term) to augment and enhance how the AI models are used.
- **How to Leverage Domain Knowledge and the Development History to Improve SE Tooling?** The research on mining software repositories (MSR) has demonstrated how knowledge from software repositories (for instance, hosted on GitHub), and communication channels, such as Stack Overflow and GitHub coding, can be leveraged to ease software evolution. These knowledge resources are being used indirectly by large language models, but there is potential to directly revisit and use text mining and summarization techniques from MSR research in combination with large language models to provide further insights that can facilitate process understanding and tool enhancements.
- **How to Capture and Leverage Developer Context to Enhance how AI improves Engineering tools?** For AI tools to provide the right support at the right time, they will need much more context about the development tasks and the developers' context — what tasks did they do before, what other tasks are they doing, what do they know, who do they know, what are their values. As researchers we need to study what context is needed, how to capture that context, and how to use that context in a way that doesn't impinge on a developer's privacy, while supporting them in a way that feels natural and allows them to feel in control and supported (we discuss these aspects more below). That is, capturing this context will further "enhance" how the AI will improve its use in software engineering tooling.
- **How to Leverage and Synthesize AI Support Across the Entire Life-cycle of Engineering Tools?** The biggest change that is mostly likely to happen is the use of integrated AI-powered assistants (such as GitHub Co-pilot) that is integrated across the entire tool chain. There is some ongoing work and progress on this direction by Tufano *et al.* where they propose an AI driven framework called "AutoDev" for the "autonomous planning and execution of intricate software engineering tasks" [161]. This paper demonstrates how the framework can

be used to auto-generate code, build, test and deploy from the Pull Request system, using multiple agents that communicate with each other. The process is invoked by and controlled by an end user. So far, it has been evaluated on simple tasks, with plans to extend to more complex tasks [161]. We expect to see more frameworks like AutoDev emerge that will rely on the orchestration of multiple agents as guided by interactions within the IDE or engineering system to support across both the inner and out loops of software development. These agents will also rely on programming language, development history and the task and developer context to act as needed. It will be important to study and understand the impacts of how those agents will make the need for some interactions from developers "obsolete" (for example, manually running tests) to introducing new interaction modes for the developers to learn and adapt to (probably through a chat interface).

### 2.3 The Impact of AI on Collaboration Practices in Software Development

*2.3.1 Trends.* Originally software developers had to work in the same room, or at least the same building, to access centralised computing hardware and its software components [138, 157, 172], as illustrated to the left of Figure 4. With the advent of distributed personal computers, collaboration and communication practices began to evolve, supporting more distributed development work [100, 138]. Video conferencing, messaging, shared repository, social media platforms and personal devices all combined in the late 90s and 2000s to enable much more flexible SE work practices [100]. Development tools began to integrate various distributed collaborative working processes and tools. Various messaging, video conferencing and co-ordination tools support a variety of asynchronous and synchronous collaborations [116, 153, 172]. More recently, AR/VR, AI-powered project management and development tools, developer agents, and collaboration with bots (bot generated content, messages, co-ordination) have led to increasing impacts on SE collaboration practices [11, 32, 111]. This includes AI-mediated collaborations, AI-support project planning and tracking, a variety of AI-powered tools, and increasingly, AI-implemented development tasks, including commit messages, comments, and notifications, shown in Figure 4 (right) [4, 50, 128].

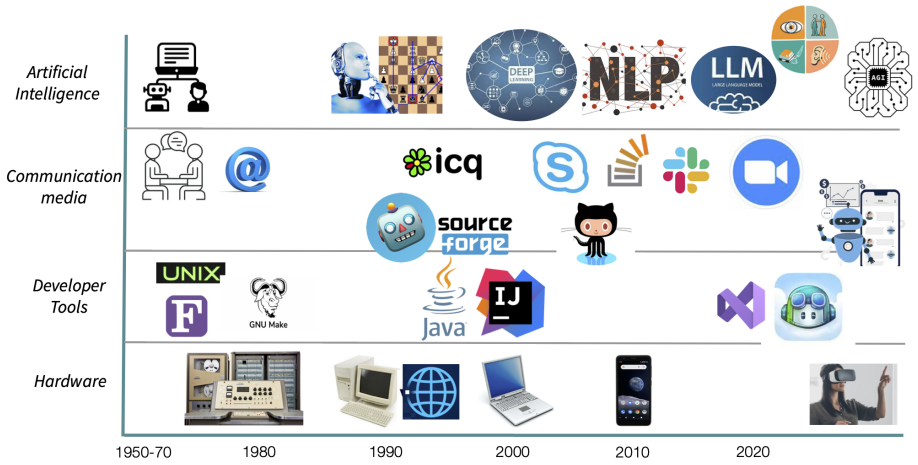



Fig. 4. Changing communication, collanoration and co-ordination practices as hardware, platforms and AI support evolve.

*2.3.2 Open Challenges.* A number of challenges emerge as we consider how software development collaboration practices may change in 2030 and beyond. Teams are increasingly hybrid, both in terms of distribution and with mixed human developer/AI-powered development agents [14, 35, 171]. Team and stakeholder dynamics will be increasingly impacted by this trend, as well as the trend to AI-powered agents and applications as software 'end users'. It is unclear how human developers and stakeholders feel about these trends [47, 162]. The nature of future communication media may undergo further changes, where significant consumers of the communications are AI-powered developer agents and not humans. How trusted, or untrusted, these significantly AI-powered developer 'team members' and developers are is yet to be determined [92]. How this will impact human collaboration and communication capabilities is also yet to be determined, not only in software development field but more widely in society.

*Nature of 'hybrid' teams:* Software teams used to be relatively homogeneous in terms of location, tooling, responsibilities and focus. Increasingly teams need to include diverse human developer capabilities, and increasingly AI-powered 'team members' – developer bots or agents [46]. A software engineering team thus includes a mix of differently skilled and experienced humans and differently capable AI-powered agents. It is unclear what impact this is having and going to have on team climate, collaboration modes, co-ordination and communication [40]. There is already evidence of significant changing practices around co-ordination and communication with bot-generated content, comments, commit messages, notifications etc [11, 128]. An interesting impact is on bot-to-bot communications. Most software development communication artefact content is still quite human-centric. Is this the right format? Is the volume appropriate for human-bot and bot-bot communication and co-ordination?

*Team and stakeholder dynamics:* AI-powered development tools and agents will impact team dynamics, but also developer-stakeholder dynamics. It is unclear how stakeholders feel about interacting with software bots to for instance to specify defects, request changes etc [46, 96]. Can an AI software development agent improve the interaction of stakeholders and hybrid software teams? How and when will end users interact with AI-powered developers [137]? In terms of empathy, can AI development agents 'understand' diverse stakeholders, their software needs, usage contexts, etc? Might there be opportunities for improved team-stakeholder collaboration when leveraging AI-powered developer agents? What is the 'right' way to use these agents in hybrid team / stakeholder collaborations? Increasingly AI-powered software systems will be 'end users' of other software systems. What does it mean to have AI as a stakeholder? How can a hybrid team collaborate with AI-powered stakeholders?

*Individual developer and stakeholder perceptions:* How do software engineers feel about the increasing nature of hybrid teams they work in [162]? Will they accept more and more AI-powered tools / developer agents? Will some collaborative aspects of software development effectively disappear as AI-powered developer agents carry them out individually or collectively without human intervention [92]? How do stakeholders feel about increasingly interacting with AI-dominated development teams? Increasingly humans have been interacting with AI-powered devices and software for instance smart home controllers, Siri-type assistants, increasingly sophisticated chatbots, ChatGPT-style agents etc. Is a natural extension to request new software features and software changes directly from AI-powered developer agents, potentially with no stakeholder-human developer direct collaboration?

*Communication media:* To date, most communication media used in software engineering - code, comments, commit messages, discussions, designs, documentation etc – assumes a human creator and a human reader. With increasing AI-powered developer agent generation and consumption of artefacts, including that for team communications, we may

want to revisit this assumption [45]. Some SE tasks that traditionally needed a form of developer communication may be wholely or mostly AI-performed tasks [1]. AI-based developer agent communication rather than human-human or even human-AI may be better accommodated in very different ways. There may be a reduction in the need for some forms of communication. For example, some human-human tasks may become wholely or very substantially AI-performed, obviating the need for substantive associated communication media. Some forms of communication may have multiple formats - human-readable and AI-optimised. The tools used to generate/create, read/consume, summarise/translate communication-related media may of course also substantially evolve, likely being heavily AI-supported.

*Ttrust and understanding:* In current SE teams it is sometimes hard to tell an AI-powered bot 'team member' from a human collaborator. This is likely to increase, and the distinction between different AI-powered developer agents, their capabilities and responsibilities, and those of human software engineers may further blur [105]. These synthetic developers that are hard (or impossible) to distinguish from human developers will be a part of the development team and be utilising the same communication channels and within the same physical/virtual spaces as human SEs. How will this impact on intra- and inter-team trust [104]? Related to the previous challenge, if communication media, contents and frequency change, how will this impact understanding of other team members, their work, their communications? The nature of SE work has already been substantially impacted by AI-powered development tools. AI-powered developer agents, their increasing deployment, the tasks and roles they perform will all impact team dynamics, professional responsibilities and collaboration styles.

*Human collaboration capabilities:* Humans have developed their collaboration capabilities over thousands of years. These have substantially changed in the 21st century with much greater remote work, efficiency and effectiveness supporting tools (especially AI-powered ones), evolving work places, rapid pace evolutionary demands on many software systems, and widespread adoption of new processes like DevOps [80, 116, 172]. In the realm of software engineering practice, developers worked originally in large rooms with substantive hardware platforms, moved to rooms with remote access terminals and personal computers, utilised open plan cubicles, worked remotely and from home, and currently often have hybrid work and collaboration contexts [88]. Each of these has brought both opportunities and challenges: freedom to work alongside others but also brought distractions; headphones and focus but with a need for controlled interruptions; synchronous video call meetings and asynchronous work, but having to manage timezones; achieving a work life balance but at a cost of living environment conflicts; and challenges in work co-ordination with physical vs virtual meetings. The COVID-19 pandemic greatly increased the practice of hyrid work for teams previous co-located [34]. With increasing AI-powered developer agents in teams, continued demands for hybrid and remote work, the impacts on human SE communication and collaboration capabilities are very important to consider [55, 156]. AI-powered software project management, communication and collaboration support, and SE work support may also both support but also challenge human SE abilities. Will virtual and augmented reality hardware be of any assistance (the metaverse for SE) [44, 149]? What will human stakeholders expect from software teams, both human and AI-powered virtual members? Might software engineering soft-skills including communication and collaboration practices be enhanced, reduced and even appraised by AI-powered tools [55]?

#### *2.3.3 Roadmap.*

- **How will increasingly common AI-powered tools and developer agents impact human-human collaboration and communication over time?** We have seen a move from collaborative spaces to headphone-dominated individual and virtual workspaces, and these have had varying impacts on human collaborative

capabilities. With the increasing use of AI, we need studies on the emerging impacts on how humans collaborate / will collaborate as they increasingly use AI-powered tools and collaborate with non-human AI-powered developer agents. This may bring some collaboration benefits, but introduces dangers of reduced capacity for effective collaboration. We need to study current and future impacts of more AI-dominated SE work, collaboration between humans mediated by AI, and in overall workplaces comprising hybrid human and AI teams.

- **How will increasingly powerful AI-powered tools and developer agents, such as bots, impact human-agent and agent-agent collaboration and communication?** Communication channels may become dominated by AI-AI developer agent and/or bot communication, some channels may need optimization for these AI-powered developer agents, and some communication artifacts may fundamentally change to better accommodate such AI-powered developer agents. We need studies that explore the impact on humans of such changes, whether multiple communication channels and artifacts need to co-exist, and how AI-powered developer agents impact SE collaboration and communication needs and practices. We need to studies to compare human-human and human-AI collaboration and communication patterns. We may need vastly different collaboration and communication platforms for AI-AI or AI-human collaborations.
- **Will AI-powered developer agents improve developer/stakeholder and developer/developer empathy or add barriers?** An under-researched and not well understood area of SE is developer/stakeholder empathy – understanding each others' different perspectives, capabilities, needs etc, both in terms of software development but also more generally as humans. Increasing our understanding of AI-powered developer agent impact on empathy is needed, as well as increasing our understanding of the importance and role of empathy between humans in SE.
- **How will increasingly powerful AI-powered tools and developer agents impact theories of distributed cognition, both human and AI-powered?** Humans and AI-powered developer agents have different capabilities, roles, communication needs, collaboration and co-ordination strategies, and will approach SE tasks with these differences. We need to develop new cognition theories about each and in combination to inform SE practices but also future human training and management and AI development and deployment.
- **Will increasing collaboration with AI-powered bots improve or decrease trust between/with humans and agents?** Many work areas have seen challenges to trust of AI-generated content, AI-performed work outputs, and AI-influenced work practices. We need to study impacts on trust between humans, humans and AI-powered tools and developer agents, and potentially also between diverse AI-powered developer agents themselves.

### 2.4 Impact of AI on the Future Software Engineering Profession

*2.4.1 Trends.* Figure 5 (left) summarises software development until recently. SE was the preserve of trained professionals who used complex software tools to engineer code and data used by end users (left). This significant software developer population needed substantial training and experience, attracted a professional software developer cohort, and was the intermediary between a large end user population and receiving software solutions. Many studies have looked into expectations of the profession, SE team leadership, professional ethics and responsibilities, recruitment and training, skills needs and gaps [5, 26, 69, 84, 102, 118, 150, 168]. This professionalism has the advantage of trained professionals responsible for complex safety and security critical software engineering. It has the disadvantages of potentially long lag times to fix defects, misunderstandings between stakeholders and software engineers, high costs, lack of sufficient engagement with diverse stakeholders, and potentially lack of specialised target domain expertise in the software team [20, 90, 127]. AI is already beginning to redefine what is software engineering, what does a software

engineer do, and even who is a software engineer [12]. As outlined above, there have been numerous changes to software tools, productivity, and to developer communication and collaboration.

As AI-powered tools continue to be adopted, their nature and incorporation into software engineering practices will greatly impact software engineering practices and both professional and citizen software engineers [25]. As the capability of these tools continues to improve, who is able to carry out (aspects of) software engineering will also arguably continue to broaden. Developing ML- and AI-powered software applications themselves is inherently quite different to more 'traditional' software systems engineering [147, 165]. Thus some key trends are emerging: the advent of "AI agents" or AI-powered bots that carry out more and more traditionally human SE tasks; the advent and growth of 'citizen software developers', which is likely to result in growth in 'citizen software engineers'; the greatly increasing demand for ML- and AI-powered software solutions across all domains; and redefinition and re-imagining of professional software engineer roles, recruitment, retention and responsibilities.

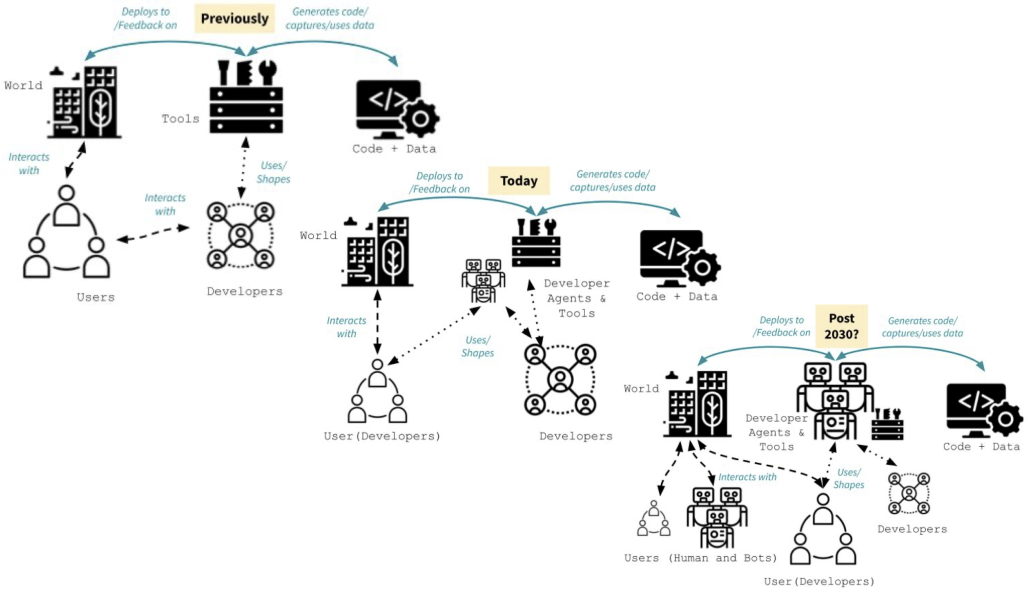



Fig. 5. Evolution of the context of software development with increasing AI-powered developer agents and AI-powered software 'end users'.

*2.4.2 Open Challenges.* There are several significant outstanding challenges of these emerging trends in software profession. Arguably chief among them for this audience is understanding the impacts on (human) professional software engineers [95]? A second major challenge is knowing how to support emerging 'citizen software engineers' – what similar but also quite different needs do they have to traditional professional engineers [12, 58, 127]? Developers are not the only role involved in software projects. Other roles include project managers, business analysts, UX designers, customer liaison etc. The impact of AI-powered software tools and developer agents on these roles are also unfolding at a rapid rate [54, 151]. A long standing challenge to the software engineering profession has been a lack of diversity – gender, age, culture, neuro-diversity, etc [139, 168]. Will AI-powered tools provide – as for increasingly citizen software engineer capabilities – a more diverse workforce [3]? Or will it exacerbate existing diversity issues? Or create new

ones [146]? Will there be such a thing as a 'professional software engineer' in a form we might recognise it in 10 years time [92]? Finally, with significant development work being done by/with AI-powered tools, that will likely further increase over time, what are the professional responsibilities going to be for (human) software engineers vs AI-powered developer agents, especially when things go wrong [147]?

*AI software engineers:* There has been a huge increase in the range and capabilities of tools. A big change is the advent of AI "software developer" agents that are becoming an increasingly important part of software teams. These AI developers will likely take on more and more responsibilities/tasks. They will likely collaborate with SEs – and increasingly with each other – to carry out a wide range of software engineering tasks [45, 46, 156]. Figure 5 (centre) outlines this trend. AI-powered tools are becoming increasingly available and adopted. Developer agents, most of which are accessible to software engineers, but increasingly also to end user developers, are becoming more common place (centre). Human developers still predominantly use SE tools to build software systems for end users. In the main, human developers invoke several AI-powered tools to carry out specific tasks. Increasingly a greater range of more powerful developer agents provide support for a wider variety of SE tasks [104]. We will move more to a model of Figure 5 (right) where a greater portion of SE work is carried out by more sophisticated AI-powered 'developers', interacting with each other to carryout out autonomously large SE activities. Increasingly software will be autonomously used by other software – end user agents – that may need and expect very different software to human users. More and more end users will be 'developers', supported by these powerful developer agents. Software using agents may themselves become developer agents to autonomously evolve software systems. This trend poses a number of challenges: what tasks do we hand over/partially hand over/not hand over? How do we check quality of work done by AI developers? How do we constrain and control development done by highly autonomous developer agents? How do we protect software development tasks being done by developer agents from both deliberate attacks and accidental errors?

*Citizen software engineers:* The emergence of low-code programming tools targeted to non-professional software developers in many domains – finance, resources, science, health, education, smart cities, etc – has resulted in the advent of the 'citizen developer' [20]. These citizen developers use these low-code platforms to carry out sometimes quite detailed and substantive software development for niche domains and projects [12, 127]. These citizen developers are the domain experts, and the low-code platforms provide much faster, accessible routes to complex software solutions than employing professional software engineers to do all of the development. With the growth of AI-powered support in low-code tools, more and more complex 'software engineering' tasks are becoming feasible for 'non-professional' software engineers. Figure 5 (right) suggests significant growth in such 'citizen software engineers', interacting with (increasingly) powerful 'AI SEs' and (less) with human SEs. The impact of these emerging AI SEs on citizen software engineering work is likely to be at least, if not more, profound and pronounced than on professional SEs [58]. Many issues remain: what can be supported by the AI SEs so that technical details are hidden from the citizen SEs; when are professional human SEs needed; and what do professional human SEs do. If SE bots and citizen developer tools continue to grow in uptake and usage, what is role of the citizen software engineer vs the professional software engineer going to look like in 5-10 years time? How do SE tasks fit into some of the emerging AI tools for other disciplines for instance scientific discovery, materials engineering, smart cities, health research and delivery, and of course the stock market.

*Other roles:* There are many other roles involved in the development and deployment of software besides just 'software engineers'. These include project managers, product managers, business analysts, UX and customer teams, operations team (even if being supplanted by DevOps), documentation experts, usability engineers, help desk operators, data

scientists and AI experts, domain experts etc. The impact of AI-powered tools and processes on all of these roles will be profound, and indeed many changes are already being witnessed. How can AI-powered project management support software development in the future [54]? What is the role of a product manager, UX designer and business analyst when many aspects of their traditional roles can be performed, to a greater or lesser degree, with AI-powered tool support [151]? Are some of these roles going to blur with citizen software engineers? When are they a distinct role vs a composite role when leveraging AI-powered tools? Increasingly we see help desk and customer support roles in many fields devolved to AI chatbots. A significant amount of software documentation is being produced by AI-powered tools. Will customers expect to interact increasingly with AI-powered 'helpers' in their applications?

*Profession diversity:* Software engineering teams have slowly become more diverse in terms of gender, age, location and cognitive differences. However, team diversity is still nowhere near end user diversity, and there continue to be many efforts to diversify the workforce [139]. This includes recruiting more diverse SE students, identifying problematic team recruitment, management and advancement practices, and identifying biases in software tools and processes. The future of AI empowered development and developnent of AI-powered software brings into the question of how AI powered tools will help enhance professional SE diversity as barriers are lowered in terms of development background [3]. As noted above, citizen or end user developers may emerge who can interact more directly with AI-powered agents or bots than those who we traditionally recruit into SE. However, AI-powered tools and AI SEs may also create new barriers to being part of the population that designs and developers software [3, 146]. This is especially a concern when we look to the future of the possibility of AI developer agents playing a critical and intensive role in software teams. Understanding the impact these tools will have on developer inclusion, productivity, experience and well-being is of paramount concern across our industry. Issues with biased AI-powered SE tools that we currently have highlight the need to carefully address this issue [166]. There are already age, gender and ethnicity biases in the SE workforce we want to address and not exacerbate.

*Job security and task responsibilities:* One emergent concern of software engineers and prospective SE students is – will I have a job in 5+ years time [92]? What SE jobs will there be and what will be their nature [95]? How will I retrain to make the best use of AI SEs and related? How are and will SE jobs be reshaped by AI-powered tools and teams [91]? It is unclear just what SE tasks will be able to be effectively given to AI-powered SEs, what kept by professional human SEs and citizen SEs, how this mix of SEs will work together, and implications on technical skills, teamwork, empathy, belonging, SE workforce diversity and inclusion, etc [141, 143]. Should we worry about the possibility that some existing developers that may be put out of work? Are there training programs we should be more proactively putting into place to reskill the SE workforce?

*Responsibility and ethics:* We have touched on the issue of addressing and not exacerbating SE workforce bias when adopting AI-powered tools and 'team-mates' [3]. Other related issues include what are ethical uses of AI-powered tools in the workplace, including data provenance and ownership of derivative works [104]? Who is responsible if an AI SE makes a mistake leading to a critical failure? How do we reason about risks, responsibilities, audits, etc in the presence of significant AI-powered SE work, perhaps across large numbers of SE agents and organisations [106, 147]? What is software quality from the perspective of engineering AI-based systems with AI-based tools [68]? How do we combat adversarial SE agents in a potentially ever increasingly complex SE environment? How should future SE teams be managed – should an AI-powered SE agent be responsible for human work and welfare? How should professional ethical frameworks be amended to accommodate AI SEs?

*2.4.3 Roadmap.*

- **What will future SEs do? Who will they work with? How will they work with AI agents? How can they best be supported by AI?** We need a variety of studies to explore emerging work practices in SE as AI-powered tools and developer agents are rolled out. This includes a rethink of traditionally human-dominated tasks, roles, responsibilities, as well as the associated collaboration, co-ordination and communication practices as outlined in the previous section. Potentially experiments with radically different approaches to SE could shed light on the future of the SE profession.
- **What can/should citizen SEs do? Not do? How can they best be supported by AI?** A traditional danger of providing non-SE expert, non-technically trained and experienced people with tools to develop software is their inability to understand, fix or even notice when things have gone wrong. Future AI-powered SE tools need to account for increasingly common citizen software 'engineers', providing them appropriate support for their lack of deep SE knowledge while leveraging their domain expertise. Studies identifying key areas of need, and indeed tasks that must remain the preserve of professional software engineers, need to be identified.
- **What are future AI-powered tools for other roles?** Everyone is looking into the impact of AI on their work, since anyone even tangentially associated with software development is impacted and will be even greater impacted. Studies need to explore emerging and needed AI-powered support for non-SE roles, their impact on these roles but also on professional software engineers they must work with.
- **Who will be responsible for failures? What is Ethical SE by an SE bot? Hybrid team?** Responsible SE with and for AI is critical. Studies must investigate root cause of diverse software and software project failure and seek to both mitigate these, but also determine appropriate responsibilities. This includes identifying unethical, too risky, unfair, and other potential negative impacts of both AI-powered SE work and AI-powered software systems.
- **How can we anticipate and prepare for bad actors as AI is adopted in Software Tools?** One of the biggest risks many software companies face are threats from bad actors that also use AI empowered tools to find novel and unexpected ways to attack software systems and steal user data. Attack vectors may be introduced at any phase of the software engineering life-cycle and thus the AI driven SE tools of the future will need to be designed with the ability to block, identify and mitigate threats in a way that is seamless across the entire tooling framework. Unfortunately the use of AI that will speed up and ease much of the friction in development tasks, may over time lead to new security threats, that will require a different type of automation to address them, forcing the design of new tools to help manage this new friction. Anticipating what AI in SE tools may "flip" or "reverse" into may help us recognize the risks and preemtively design tools to mitigate these concerns.

### 2.5 Impact on Software Engineering Education

*2.5.1 Trends.* We have built up over several decades the foundations of a software profession, including initial education and on the job training. The Software Body of Knowledge (SWEBOK) has been refined over many years by ACM and IEEE to try and guide us in ensuring well-trained software engineers are available to society [22]. We have used this body of knowledge to construct higher education and workforce training for professional software engineers and to better understand their skills gaps and how to address them [60]. Our curricula and body of knowledge have undergone periodic updates to incorporate a number of technical and professional changes. These include for instance adoption

of agile methods, cloud platforms, DevOps, security and privacy, codifying various professional responsibilities, and incorporation of more team- and project-based learning approaches.

*2.5.2 Open Challenges.* A number of professional education challenges present. What should the SWEBOK contain now? In 5 years time? What does the emergence of AI-powered software tools, citizen software engineers, developer software agents, and AI-powered software 'end users' mean for software education [89, 107]? What are the future skills needed by professional software engineers, citizen software engineers, other roles involved in SE, and indeed software development agents [30, 54, 95, 151]?

*SWEBOK:* It has been a challenge to keep the software body of knowledge up to date as the technical, professional and societal demands on the profession have changed. As discussed in the previous subsections, it is difficult to foresee just what the change in software engineer tools, collaborations, professional skills and expectations and roles will actually be. With the advent and increase in AI-powered software tools, just what skills does a software engineer of the near and more distant future actually need? It may be that new processes, tools and techniuqes, supported by AI-powered tools and developer agents, substantially change the nature of software engineering and mean some traditional skillsd are not required, or much less commonly required. The ability of domain experts and end users – citizen software engineers – to develop substantial aspects of software systems may mean the number and nature of skills of professional software engineers substantially change. Should there be a SWEBOK for citizen software engineers? For AI-powered developer software agents? It seems essential for software engineering students to gain knowledge of engineering AI-powered software systems, including the various technologies involved [29, 141, 143].

*Training for work in hybrid development teams:* Software engineers increasingly work with AI-powered tools, collections of AI-powered tools, and developer agents. They will continue to work with humans, including other software engineers, other development-related roles and end users and stakeholders. However, the expectations, tasks and roles of both other humans and AI-powered agents and tools may greatly change. It is unclear how a hybrid development team of humans and AI-powered developer agents should be formed, organised, managed, tracked etc. It is unclear what the impact on human software engineers will be of their changing roles and expectations, heavy use of AI-powered tools and collaboration with AI-powered software agents, hybrid and remote work, and how changing roles of other humans involved in development may impact professional software engineers [27]. At the least, it seems essential that software engineering students will need to gain significant experience and skills with choosing, configuring, using and assessing a wide variety of current and emerging AI-powered tools and 'team mates' [29, 141]. They also need experience in building a variety of ML- and AI-powered systems [86, 165] and carefully addressing ethical and responsible AI issues [130]. They may also need to gain a better understanding and experience with changing expectations of their stakeholders who themselves are becoming much more familiar with and much heavier users of AI-powered software in their own work domains. AI-mediated stakeholder interaction seems likely to continue to increase. As noted in earlier subsections, AI-mediated developer interaction also seems likely to continue to increase. Training and experience in such environments is likely to be essential for future software engineers.

*Software engineering education delivery:* Like all other educators, software engineering educators are carefully considering how to leverage AI in their teaching and assessment, both in a higher education setting and in the workplace [133]. AI-powered tutors and assessors have been experimented with in many domains. Many outstanding issues remain, but it seems highly likely that software engineering education delivery itself will be highly impacted by AI-powered tools [53].

Some challenges we are familiar with for instance did the student write their code/tests/design or was it copied/adapted (or produced by generative AI)? The SE and CS education community has deployed various AI-powered tools including code and other software artefact plagiarism detectors for many years. Many software engineering programmes have been experimenting with incorporating generative AI and other techniques into curricula [37, 53, 76, 133]. Thus deploying AI-powered software tools and developer agents during software engineer training does not seem a big stretch. However, assessment practices are likely to need a significant rethinking. Software engineers have been heavy adopters of generative AI in supporting on the job training, technical question answering, generation of tutorials and other supporting materials for learning.

*Attracting and retaining future software engineers:* Given the dramatic changes likely in the software engineering profession, including much traditional work being taken up by AI-powered tools and developer agents, many are wondering just what the future profession will look like – and indeed if there will be one [92]. Increasing questions at University open days around what the future careers in software engineering will look like, what career trajectory they will bring, and whether it is going to be a good career choice demand responses. It may be that many aspects of software engineering that attracted people to the professional no longer exist or are massively changed in the near, let alone distant, future. It may be that a very different set of skills and aptitude will make for a 'good software engineer' of the future [102]. How will we 'sell' the prospect of software engineering and being a software engineer? Will there be vastly different pathways into software engineering and career pathways within the profession? Will the need for the traditional concept of the professional software engineer increase, decrease or remain largely the same with the continued development of AI-powered developer agents. Will domain knowledge be far more important than software engineering skills and knowledge, with much of software engineering undertaken by developer agents under the instruction of end users and citizen software engineers? Or will the increasingly complexity, range and capability of software systems necessitate a continued substantive profession?

#### *2.5.3 Roadmap.*

- **What should be in the Software Body of Knowledge in 2030?** We need to engage with practitioners and educators to reformulate the SWEBOK cognisant of the big impact AI-powered tools and developer agents are having in SE work and education. The degree of change in emphasis on topics in SE curricula and for different future SE roles also needs careful consideration.
- **What does a SE curriculum and an SE career of the future look like?** A danger of all education curricula is backwards-looking or current skill demand driven design, instead of future likely need. Educating all future workers for better awareness and skill in working with emerging AI-powered technologies is essential, including their current and likely limitations. Significant experience with diverse current and emerging AI-powered tools when educating software engineers seems critical. Is the curriculum similar or quite different for a 'citizen software engineer' to that of a professional software engineer curricula? The needs of non-professional citizen software engineers must not be ignored and their education may need to be quite different to future professional software engineers. For start, they are likely to need a primary domain-specific education with engineering software leveraging this domain expertise a secondary focus, however well-integrated. On the job citizen SE training (perhaps significantly AI-enabled) may be more viable in many instances.
- **How do we effectively deliver a future SE curricula, including leveraging AI educational tools and providing experiential learning and practice with AI SEs?** While SE education has always had to be a degree

process, technique and especially tool agnostic – all organisations have greater or lesser degree of variation in each of these – providing knowledge, training and experience with some practical combination of these has been a feature of almost all SE degree programmes for many years. However, many software engineers have gained significant skill enhancement via tutorials, workshops, video tutorials, online communities and – increasingly – AI-supported guidance and support. It seems likely that further AI-supported education and training studies are needed, including emerging AI-supported approaches to delivery and assessment.

- **How do we attract and retain excellent software engineers of the future?** There are promising and threatening aspects to greater use of AI-powered tools and emergence of AI-powered developer agents. Studies need to include a diversity, inclusion and equity aspect to adopting AI-powered SE. This includes addressing bias and unfairness in AI training, diverse usage practices, diverse human SE needs and expectations, and carefully assessing negative consequences or potential consequences of adopting varied AI-powered solutions. It also remains to be seen if the emergence of AI-powered tools that significantly change SE work and SE job expectations enable many who previously did not embark on an SE career to do so. Proactive efforts to present varied SE work of the present and of the likely future, engagement with potential future students and professionals, and increasingly engagement with likely future citizen software engineers and part-time professional software engineers are needed. Ultimately we need to include risk assessments of AI-powered solutions from individual, team, organisational and societal perspectives.

## 3 HUMAN-CENTRIC SOFTWARE SYSTEMS IN THE ERA OF AI

Software systems are inherently designed for humans, who interact with them either directly, such as with software applications, or indirectly, as with embedded systems. Over the years, advancements in technology have transformed these interactions. From mice and GUIs in the 1970s to the internet and web in the 1980s, mobile devices and Wi-Fi in the 1990s, and cloud computing at the turn of the century, technological shifts have continuously redefined human-computer interaction. More recently, extended reality, wearable devices, and generative AI exemplify the innovations driving these changes.

The rapid evolution of extended reality and generative AI is fundamentally altering human-software interactions. The blurring lines between virtual and actual reality, coupled with the rise of voice, video, and gesture-based interactions, are gradually replacing traditional GUI-based interfaces. Generative AI further amplifies this shift, reshaping the role of humans from *passive users* to *active participants* in smart, human-centric systems and ecosystems. These ecosystems range from smart living applications, such as smart homes, smart grids, smart transportation systems, to large heterogeneous *human-centric ecosystems* that emerge from the coexistence of systems and humans in complex environments.

However, human diversity —encompassing personality, emotions, ethics, culture, age, and gender —significantly influences how individuals interact with AI-powered systems. These interactions, enabled by generative AI, extended reality, and multimodal communication methods, redefine not only software design but also the way humans live and function within software ecosystems.

We define *human-centered systems* as software systems designed and operated based on human values and needs. These systems, often highly automated and AI-driven, aim to balance automation with the preservation of human control. They strive to deliver ethical, efficient, and inclusive outcomes, maximizing the benefit to humanity. As illustrated in Figure 6, McLuhan's Tetrad (introduced in Section 1) offers a framework to analyze the disruptive impact of these systems on software engineering by examining what they enhance, retrieve, reverse, and obsolesce.

Human-centric systems *enhance* diversification, customization, evolution and adaptation, and *obsolesce* rigid and static operations. They enhance human diversity, heterogeneity, evolutionary, self-adaptive and autonomic software engineering activities, field testing, autonomous bots and data productization. They obsolesce rigid and recurrent engineering activities, like structural and regression testing, the MAPE cycle, and component-based architectures. Human-centric systems *reverse* human-computer interactions and *retrieve* requirement engineering and dynamic analysis. They reverse natural language, image and video processing, generative AI, and extended reality, which become the cornerstones of the new role of humans in the system. They retrieve run-time simulation and digital twins to reason about seamlessly evolving behavior that emerge from evolving human interactions. At the same time they *retrieve* models at runtime, dynamic and incremental analysis, to reason about evolving behavior that derive from unconstrained interactions, and domain specific languages, requirement engineering and enterprise architectures to drive Generative AI towards acceptable solutions.

The four diamonds of Figure 6 outline what smart- human centric ecosystems enhance, retrieve, reverse and obsolesce. The ENHANCES diamond indicates what smart ecosystems largely amplify: Diversity and customization, self adaptation and evolution, field testing and autonomous bots, autonomic verification, human models, and data productization. Customized services and systems have been studies in the last decades. Software product lines [31] and APIs [98] are two opposite ways of customizing software systems by tailoring general frameworks to specific needs (product lines) and opening systems interactions towards extensible frameworks (APIs). Generative AI and extended reality lead to ecosystems that autonomously emerge, evolve and adapt over time towards ultra large, smart, human centric ecosystems. Self-adaptive and evolving systems have been widely studied since the beginning of the century [59] to cope with the lack of central control and specifications of Ultra-Large Software Systems. The intrinsically evolving nature of smart systems and ecosystems emphasizes the need of self-adaptive approaches to cope with emergent behaviors and sometimes unavoidable failures. Field testing [19], self-adaptive systems [33], autonomic verification, and more recently infrastructure testing and test bots [46] move testing and verification from testbed to productization environments, and pave the way to a completely new way of verifying adaptive and evolving smart human-centric ecosystems.

The RETRIEVES diamond highlights technologies that have been widely studied and used in the past and that will find a new role in smart, human-centric ecosystems. Simulation [13] and digital twins [145] are commonly used to train humans with safety-critical systems and test complex cyberphysical systems. Testing the evolving and adaptive behavior

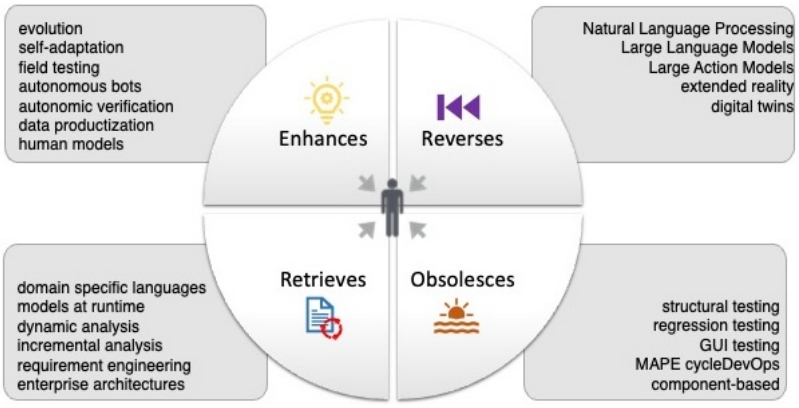



Fig. 6. The disruptive effects of human-centric systems

of smart, human-centric systems requires complete and consistent digital twins to capture the all and only aspect of the real system that are necessary to observe the behavior of the real system. Formal models, static and dynamic analysis have been studied for many decades with many successful applications in safety critical systems. Smart, human-centric ecosystems will retrieve dynamic and incremental analysis to cope with their evolving and adaptive behavior. Smart, human, centric ecosystems will retrieve domain-specific languages [112] and enterprise architectures [140] to cope with the dynamic complexity of the systems.

The REVERSES diamond indicates the technologies that smart, human-centric ecosystems will flip into when pushed to the extremes: natural language, voice, image and video processing, and more generally generative AI and extended reality, all of which are the fuel of the future smart ecosystems.

The OBSOLESCES diamond summarizes some popular engineering approaches that cannot cope with the new characteristics of smart systems and ecosystems: structural, regression and GUI testing, the MAPE, Monitoring, Analysis, Planning and Execution autonomic cycle, and in general self-adaptive cycles. Structural, regression, and in generate common testing approaches, including GUI testing, assume both the repeatability of the executions of the target systems executed on testbed platforms and the availability of some specifications in the form of test oracles to validate the results. GUI testing assumes that users interact with the system only through the GUI. Smart, human-centric ecosystems evolve over time, adapt to emerging scenarios, and depend on unconstrained human behaviors that cannot be fully reproduced on the testbed.

The subsequent sections examine the impact of emerging technologies on human-system interactions. These technologies, which drive the development of human-centric systems, enable continuously adaptive software that challenges traditional user interface design, fosters the spontaneous emergence of large ecosystems from the implicit interactions of independently managed systems, and reshapes development and operational practices. In particular, Section 3.1 explores the evolution from traditional GUIs to intelligent AUI (Adaptive User Interfaces), Section 3.2 addresses the challenges of designing and verifying large, emergent ecosystems, and Section 3.3 examines the shift from NoOps to AIOps in development and operational activities. Together, these discussions highlight how new technologies are driving the creation of continuously adaptive, human-centric systems that redefine the interaction between humans and software systems.

### 3.1 From GUI to Intelligent AUI

*3.1.1 Trends.* The demand for efficient and user-friendly software systems is rapidly increasing. Modern software systems must continuously adapt to the unique characteristics of diverse users, platforms, and environments. This shift forces developers to embrace change as an integral part of the development process. The notion of *software adaptation*, which currently focuses primarily on system-level changes, is evolving towards *intelligent adaptation*, where AI, software systems, and humans collaborate to continuously adapt and evolve human-centric ecosystems in real-time, enhancing their quality and user experience. In this context, we envision humans and AI-enabled systems collaborating to achieve common adaptation goals over time, *evolve* together, *learn* from each other, and collectively *improve*.

Since diverse humans are involved, we need *intelligent adaptations*, where human-centric smart ecosystems can learn to adapt themselves to different users based on the characteristics of the context of use. This includes characteristics of the *human being*, such as emotion, technical proficiency, gender, personality, physical and mental impairments, the *platform*, such as Web, desktop, tablet, phone, mobile applications, and the *environment*, such as location, mobile conditions, light, noise level, physical configuration, organisational and psycho-social constraints, as well as available

interaction data. Self-adaptive systems, model-driven engineering, and AI are key enabling technologies to support such intelligent adaptations, although significant challenges remain.

*3.1.2 Open challenges.* Humans come from diverse backgrounds with varying expertise, cognitive, and physical traits. Their needs and preferences may change over time, requiring software systems to identify new scenarios and adapt appropriately without costly re-engineering. *Adaptive User Interfaces* (*AUIs*) can address diverse user needs by customizing the user interface to meet individual needs, goals, and context of use [144] [121]. While the concept of UAI is not new, advancements in AI techniques bring new perspectives on its realization. Here, we analyse the current status of this concept and highlight the main open challenges.

*UI adaptation in HCI and SE:* An effective AUI must apply the right adaptation at the right time in the right place to maximize its value for humans. UI adaptation typically involves a three-stage process: *perception* (recognizing and interpreting the current context), *decision* (determining whether an adaptation is needed and what action to take), and *action* (implementing the adaptation). This process can radically change when a third party, such as an AI agent, supports the adaptation process (i.e., to recommend or perform UI changes during user interactions with the system).

Over the last three decades, the Software Engineering (SE) [7] [8] [174] [6] [173] [167] and the Human-Computer Interaction (HCI) [158] [99] [64] [163] [113] [176][63] communities have made substantial progress in AUIs' design and evolution, particularly leveraging AI techniques. However, emerging technologies, especially Generative AI, introduce new challenges and opportunities. The HCI community has proposed numerous methods for UI adaptation, but most focus on the adaptation of individual user interface elements (e.g., menus) [66] [163] [158]. There is a pressing need for holistic approaches that address the adaptation of the entire user interface in the context of human-centric smart ecosystems. The emerging field of Intelligent User Interfaces (IUIs) explores the interplay between AI and HCI. While many IUI techniques focus on improving communication between users and systems through advanced interaction methods —such as natural language processing, gaze tracking, and facial recognition— not all IUIs have true learning or problem-solving capabilities. The SE community has introduced principles and technologies to support UI adaptation at the system-side (e.g., MAPE-K adaptation loop, models at runtime), but often relegates the perception, decision and action stages of adaptation on the end-user side, typically addressed in HCI, to a secondary role.

*Limitations of current AUI approaches:* Despite some promising empirical studies [160] [39] [56] [176], current AUI approaches remain unpredictable (users do not know when and how adaptations will take place) [57], choose detrimental adaptations with unacceptable frequency, incorrectly capture user needs, cause cognitive disruption (users are disrupted by the adaptation) [101], lack user involvement (users cannot actively participate in the adaptation process), and lack explanation (users are not informed of the reasons for adaptation).

Adaptive strategies for AUIs open many challenges to effectively leverage learning capabilities: how to select an adaptation strategy; how to model human factors; how to model AUIs at runtime; how to coordinate human and technical aspects; how to handle heterogeneous input data; how to predict user experience with AUIs; how to support decision making; how to support AUIs in specific environments such as extended reality.

*Selecting adaptations:* A key technical challenge of AUIs is how to select adaptations, and design an effective adaptive user interface in the presence of a huge design space of adaptive user interfaces [163]. A system must decide *what* to adapt, and *when* or *when* not to change. An important issue is to determine the *sequence of UI adaptations* to be

carried out to maximize the user benefits and reduce the costs, for instance, improve the system quality, improve the user performance or experience, reduce the workload.

Several studies have applied different approaches: heuristics, rules and Machine Learning (ML) methods (supervised learning, reinforcement learning) to support the adaptation process. *Heuristic-based* approaches adapt specific UI elements [38], and work well when the sensed input is highly predictive of the most appropriate adaptation, for example, in the context of menu-based interaction, heuristics that exploit click frequency, visit duration or other metrics obtained from interaction data are frequently used. *Rule-based* approaches modify the UI behaviour based on pre-defined rules, like, event-condition-action rules [8]. They transparently monitor the executed adaptation actions, and thus facilitate the modification of the adaptation process, however, they do not support emerging behaviours, and require experts to specify the user goals. *Predictive ML-based* approaches learn how to adapt AUIs. *Supervised learning approaches* work well when the user state is highly predictive of the appropriate adaptation, for instance, a mapping is learned between user data and suitable adaptations, but require deep training with high-quality datasets that describes the consequences of possible adaptations on potential users. *Reinforcement Learning approaches* formulate the UI adaptation process as a stochastic sequential decision problem, and train agents based on some rewards received from the environment for certain actions. They require an extensive amount of poor attempts to learn a good policy, and this is a strong limitation for large state-action spaces, such as those needed to design AUIs for human-centric ecosystems. *Model-based* and *multi-agent* reinforcement learning are promising approaches to address these limitations. Todi et al.'s model-based reinforcement learning approach uses HCI models to predict rewards for each state during simulations [158]. An HCI model is a computational model that explain how users interact with interfaces at the level of individual human cognition [131]. Langerak et al.'s multi-agent reinforcement learning approach pairs a user agent that mimics a real user, and learns how to interact with a UI [99] with an interface agent that learns UI adaptations to maximize the user agent's performance. Model-based and multi-agent reinforcement learning indicate interesting research directions that need to address their limited scalability and an effective way to determine the rewards [61].

*Designing intelligent AUIs:* The design of intelligent AUIs raises several key challenges about how to structure software and AI components to support intelligent UI adaptation, incorporate human feedback when training AI models to support UI adaptation, and support the human-centric adaptation. Abrahão et al. [2] recently proposed a conceptual reference framework to support model-based intelligent user interface adaptation, but we sill need to experiment with different AI techniques in different contexts of use, to structure software and AI components to support intelligent UI adaptation. The current approaches based on predictive HCI models to incorporate human feedback when training AI models do not take into account changes in human behavior that may occur during user interaction with the system [158] .

*Modeling human factors:* Many approaches rely on some user models to support the adaptation of human-centric ecosystems. However, structure and usage of the current user models are not sufficiently explicit to help developers model human factors and implement UI adaptations. There is a distinguished body of work looking at how best to take users' existing practices into account when designing software. These include consideration of the social and organisational context, designing software where there is a cognitive fit between users' mental models and software representations, acknowledging that users have cognitive biases that will affect system adoption. Participatory Design and Co-design [142] have tackled these challenges head-on by including users and other stakeholders as equal partners from the very beginning of the design process. However, the research on fully and descriptively modelling the users

has not produced fully satisfactory results so far. It is also unclear how models representing human factors can be used to drive UI adaptations. Therefore, modeling and design of human factors are essential steps toward designing and adapting human-centric ecosystems.

*Runtime models of AUIs:* Models at runtime [21] have been successfully used to automatically reflect changes from a system into changes in models, and vice versa, in several domains. The availability of configurable run-time models makes it possible to use them as interfaces for monitoring and adapting AUIs, and to test adaptations at the level of models before actually adapting the running system [33]. However, the traditional adaptation in which the variability is fully pre-defined does not adequately address human diversity. We need smart UIs that can learn how to adapt to different users based on a user model that captures the user preferences, behaviour, style of interaction, expertise, emotions, etc.

*Diversity in input user data:* We can decide how to adapt UIs based on a variety of sources including empirical data, end user feedback, and biometric sensors that provide essential data to recognize users' physiological states during the interaction in various scenarios. For example, we can detect emotions by processing data from electroencephalography (EEG) sensors [65] [62]; We can measure stress and arousal with data from the skin conductivity monitored with galvanic skin response (GSR) sensors [124]; We can measure stress levels and muscle-arousing activities with electromyographic (EMG) sensors [71]. We see many different physical signals that we can use to support UI adaptation more effectively. However, how can AI agents learn from this data to recommend more meaningful UI adaptations is a largely open issue.

*Predicting user experience with AUIs:* We can measure some aspects of User eXperience (UX) by analysing physiological measures that represent different human affective states (for instance pleasure, stress, relaxation) when performing a particular task. These kinds of measures have received increasing attention from the SE community in the last few years. A recent systematic literature review on the use of physiological measures in SE [169] revealed several empirical studies on measuring the cognitive load in SE activities, such as code comprehension, code inspection, programming, and bug fixing. However, there are only few experiments on understanding end user's emotions and feelings when interacting with user interfaces and human-centric ecosystems [62]. We can measure and monitor end user emotions and feelings during user interaction to improve the user experience.

*Trade-offs in decision making:* There is a lack of decision-making processes that support trade-offs between system quality attributes, user characteristics and adaptation rules. We need guidelines to define and operationalize decision-making methodologies, and to guide decisions on the use of AI inferences (for instance, exploring different ML algorithms) instead of simply waiting for user input.

*AUIs in extended reality environments:* EXtended reality (XR) is a disruptive computing platform that merges virtual (VR), augmented (AR), and mixed reality (MR) towards computing environments where end users seamlessly interact with both the real and virtual world, with great potential to a variety of sectors, for instance, entertainment, education, medicine, manufacturing industry. However, we still have little experience on designing AUIs that present interactive content to end users in XR immersive environments. XR requires a user interface that overlays virtual content onto or embeds it within the real, physical environment without costs to end users' attention, effort, and safety [159].

#### *3.1.3 2030 Roadmap.*

- **How can human diversity and different human needs drive UI adaptation?** Humans are very diverse in needs, preferences, values, gender, age, ethnicity, culture, language, educational attainment, socio-economic status, living and work conditions, personality, motivation, emotions, behavior, cognitive challenges, physical challenges, and so on. Each human is unique and has unique needs, preferences and ways of interacting. The human differences dramatically impact on the design and adaption of AUIs: how do we model an infinitely large space of diverse characteristics, and use them to drive UI adaptations? What are the best abstractions to effectively represent the human characteristics? What is the most suitable theory to capture them? Digital twins and domain specific languages offer great potential to represent and simulate human characteristics and their interaction, but we still need to formalize these diverse human characteristics and their impact on AUIs, by leveraging and/or adapting existing theories in fields such as social sciences, behavioural sciences, cognitive psychology, etc. The area of individual differences in psychology has a long track record of validated measures of stable individual traits over time (see for instance [132]). From a more practical viewpoint, a theory for human diversity in AUIs can give us input for decision-making regarding choices of UI adaptations and their impact on user characteristics and system quality.
- **How will AI-powered systems (AI Agents) interact effectively with humans?** We need precise design guidelines for human-AI interactions. Recently, 18 generally applicable design guidelines for human-AI interaction have been proposed [9] [103]. Examples of such guidelines include: Make clear what the system can do; Show contextually relevant information; Learn from user behaviour and update and adapt cautiously. While these guidelines provide a starting point for understanding human-AI interactions, it is not clear how they can be operationalized and used to design intelligent AUIs. How can the system learn from user interactions with the system and AI agents to recommend meaningful adaptations? AUIs should integrate adaptation capabilities and computational learning. Adaptation could be accomplished by using the knowledge stored in a User Model and inferring new knowledge using the current user interaction data. Computational learning and models at runtime can be used when knowledge stored in the User Model is changed to reflect new situations or runtime data. However, how do we represent a User Model, use it at runtime, and evolve this model to reflect the changes in the user needs and preferences?
- **How can we design Explainable AUIs?** A large body of work exists and continues to grow on how to increase transparency or explain the behaviors of AI systems (for instance, [93] [135] [10]). However we need techniques and mechanisms to explain the underlying rationale of an intelligent AUI to the end user in a logical and understandable way. The main challenges in explaining UI adaptations made by AI agents are: Shall we explain the individual decision or the whole policy? How can we actively measure users' understanding of the agent's decision process? How to present the explanations to end users (for instance, white box vs. black box explanations)? How to get users' feedback to continuously improve the explanation process? How to provide users with easy controls over the output of the intelligent AUI/ML algorithms?
- **How can we create an accurate mental model of the ML algorithms underlying AUIs?** We need transparent AUIs with respect to the user's actions that can contribute to the output of the adaptation algorithm. We need to understand how people form mental models of the way the AI agent works, given different types of eXplainable Artificial Intelligence (XAI) explanations, and the cognitive loads they incur in forming these mental models. Few recent studies started investigating this problem [10], however, we still do not understand how to gather empirical evidence for designing explainable AUIs?

- **How can we design AUIs that adhere to fair-by-design principles?** Users may experience unexpected or confusing information that violates the expectations and hints at algorithmic bias, while interacting with the system [48]. How can we handle bias in the UI adaptation process? How can we detect bias in the datasets or decision-making process supported by AI agents, for example, wrong decisions to adapt the UI taking into account the user's gender or membership of marginalized social groups (race, health, disadvantaged socio-economic background)? UI designers need specific training and techniques to recognize social norms and biases. GenderMag [24], a method for identifying gender biases in user interfaces, only scratches the surface of a huge problem that need further investigation. We need tools for assess and repair fairness; We need controlled experimentation environments to promote awareness of fairness during the AUI design, and effectively test and improve bias-related issues before releasing an intelligent adaptation strategy for AUIs. The AI Act, a proposal for a European regulation on AI released in April 2021, defines the concept of *sandboxes*: controlled environments that facilitate the development, testing, and validation of innovative AI systems for a limited time before their placement on the market, a first step in this direction.

### 3.2 From Systems-of-Systems to Human-Centric Smart EcoSystems

*3.2.1 Trends.* The key role of humans in software-based systems has been acknowledged and studied since the early 2000. Common software engineering systems constrain human-system interactions within well defined interfaces that restrict the human actions that software engineers consider in the software development process. The software process models assume the existence of requirements to elicit and analyse, and indicate how to model and implement the system architecture, and how to verify and validate the system with respect to its requirements. Common approaches to design and validate software systems consider humans as mere *users* of the system, and rely primarily on system models, thus largely ignoring the impact on human attitude on the ecosystem. Northtop et al. recognized the new role of humans in Ultra Large Systems, ULS, already at the beginning of the century (*"People will not just be users of a ULS system; they will be elements of the system"* [122]). More recently, John Grundy has emphasized the role of humans in *human-centric systems* that heavily rely on the *use of Internet of Things-type sensors, interactors, controllers, .... capture and use diverse types and amounts of data, ...* and *... increasingly require some hybrid form of cloud- and edge-computing*, and stresses the impact of *human-centric issues* [73].

The technological revolution of the last two decades and the recent advances in Generative AI, augmented reality, and the internet of things have a disruptive impact on human-centric systems. El Noussa et al. characterize human-centric systems that heavily rely on Generative AI as *Smart EcoSystems (SES), multifaceted systems that emerge from the composition of independently-operated and autonomous systems with smart functionalities, and that evolve over time, as in the case of next-generation smart cities, smart buildings, smart grids, and more generally a future smart planet.* [42]. Smart, human-centric ecosystems, like smart cities and smart grids, evolve and adapt over time to match to the infinitely many human activities and needs that heavily impact on the ecosystem. New types of independently-operated smart systems, like autonomous vehicles and mobile devices, constantly enter and exit the ecosystem. GAI-centered systems learn from the ecosystem, and adapt their behavior accordingly. The human behavior changes and evolves within a smart, human-centric ecosystem, and determines new quality attributes. While classic systems operate according to well understood requirements that determine the correctness of the system, *smart systems*, like bots and autonomous vehicles, challenge design and verification with adaptive behaviors that emerge and evolve over time to adapt to different human behaviors. *Smart, human-centric ecosystems*, like smart grids and smart cities, are not owned, specified and maintained by a single owner. The city authority regulates the behavior of the different systems within the city,

however, the city authority does not own the many systems that coexist in the city and that evolve according to specific and sometime intrinsically contradicting requirements [41].

*3.2.2 Open Challenges.* Many open challenges emerge when trying to engineer future smart, human-centric ecosystems. Traditional ideas of correctness need to be revisited. Humans are diverse and interactions may be unpredictable, unconstrained and behaviours unanticipated, especially as systems become more self-adaptive. Traditional software engineering methods, including testing, analysis, and monitoring, all struggle with aspects of smart ecosystems engineering. Self-adaptive systems are likely to grow significantly. It is unclear how humans impact such self-adaptation and how various AI-powered solutions will support diverse self-adaptive smart ecosystems.

*Correctness of a smart, human-centric ecosystem:* The concept of correctness – defined as compliance with a specification – does not sufficiently capture the quality aspects of a smart, human-centric ecosystem. The behavior of a smart city can obey the regulations and can well suit the needs of the humans in the city, however, it is not simply correct or wrong. We may witness a major degradation of a smart city, but we do not have the elements to define a failure as a deviation from the specifications, in the absence of complete and agreed specifications of the ecosystem and in the presence of emerging and sometime contradicting behaviors of the systems in the ecosystem. For example, we can observe a major degradation of the smart city due to unexpected and implicit interactions among the traffic control system that reduces the access to the highway to avoid traffic jam, the pollution control system that reduces the speed limits to lower the pollution, and the heterogeneous and unpredictable behavior of the autonomous and non-autonomous vehicles. Noura et al. propose a new concept of *healthiness* to capture the acceptable behavior of a smart ecosystem that capture the strength of the ecosystem, the ability to exhibit a stable behavior, and the capability to react to unavoidable crises [42], a preliminary step towards a new concept of quality of smart human-centric ecosystems.

*Unconstrained human-ecosystem interactions:* Unconstrained human-system interactions within smart, human-centric ecosystems upset the software engineering landscape. Software engineers cannot engineer smart ecosystems by limiting their attention to the system interfaces while ignoring the human attitude. Autonomous vehicles cannot simply return the control to the drivers in the case of unexpected scenarios. Software engineering shall take into account the infinitely many possible responses and reaction time of drivers with different attitude, personality and expertise. When engineering smart ecosystems, software engineers cannot simply rely on an expected uniform human behavior according to regulations. They must consider the different behavior of diverse humans with different motivations, personalities, emotions, languages, and more.

The *greedy crash scenario* presented in [42] is a good example of how human motivations and personality may lead to a smart ecosystem failure also when the systems that comprise the ecosystem behave correctly: An unexpected event, like a fire alarm in a busy underground station, may cause a high peak of requests of both taxi and Uber rides in the smart city ecosystem. Uber raises the fares in the presence of unbalanced requests and available drivers, and the tests can verify that the ecosystem restores the usual request-availability balance with common pricing after a relatively short perturbation, when humans behave as expected. However, some drivers, the *greedy drivers* as called in [42], may decline the requests, by expecting further increases of fares. A few greedy drivers may not impact the ecosystem, however a test suite that takes into account also the human behavior, and in particular possible *greedy behavior*, can reveal that an unpredictably large percentage of greedy drivers can lead to an explosion of fares and a consequent failure of the ecosystem, even if all systems that comprise the ecosystem behave correctly [42]. A *greedy behavior* can depend on motivations and personality. Conscientious drivers, who rely on Uber as their main income,

hardly decline requests, while overconfident drivers, who rely on uber for some extra income, may exhibit a completely different behavior.

In general, approaches to assess the quality of smart cities must take into account the attitude and personality of humans and social groups that may lead to different behaviors and change over time due to unexpected implicit interactions of smart systems with the smart city. Human behavior has been widely studied for centuries, and there are many modes of different aspects of human behavior. The *Big Five personality* traits, also known as the *OCEAN* model [125, 126], that behavioral psychologists have developed over the last decades well captures the human personality, and can greatly help software engineers understand the complex intertwining among human actions that stem from different personality and characteristics, to define suitable models for engineering smart, human centric ecosystems.

*Self-adaptive smart ecosystems:* Smart systems, like self driving cars, rely on deep learning engines, the behavior of which depends on the training data and non-deterministically evolves over time. As such, smart ecosystems cannot be either completely specified in advance or deployed on a testbed platform, and their behavior is seldom repeatable. Human are active elements of smart ecosystems and their interactions cannot be framed with user interfaces. The classic MAPE, Monitoring, Analysis, Planning and Execution autonomic cycle, and in general traditional self adaptive cycles, monitor the autonomic system and execute corrective actions. The main approaches defined in the last decades follow the *closed-loop approach* postulated by Müller et al. *"Architectures based on closed-loop feedback control offer appropriate capabilities to address this challenge"* [117], and focuses on the cyberphysical system. Smart ecosystems include humans as key elements that escape monitoring and executions in the forms required for MAPE self-adaptive cycles. The next generation of self adaptive approaches cannot ignore humans. We need new self adaptive approaches that include models of both the systems and the humans, who comprise the ecosystem. Smart human-centric ecosystems enhance self adaptive and evolving systems with models of the interdependence of human actions due the human characteristics. New models of human behavior will complement autonomic verification to verify the complex intertwining between the smart systems and ecosystems and the humans.

*AI-powered testing:* Most testing approaches execute tests cases on testbeds, before deployment. Static analysis approaches check code and artifacts. Dynamic program analysis work on execution traces. Smart, human-centric ecosystems emerge and evolve over time with continuously updating training of AI-powered systems, and systems as well as humans continuously entering and leaving the smart, human-centric ecosystem. The intrinsic evolving and adaptive nature of smart ecosystems makes it impossible to fully test the ecosystem on testbed. Field testing [19] and more recently test bots [46] have been studied since the beginning of the century to cope with failures in production. Both the *in-vivo* and *in-vitro* approaches developed so far aim to test software systems with data from the production environment, and target mostly unit testing. Smart, human-centric ecosystems enhance field testing as the primary way to both assess the quality of evolving behavior, and capture emerging failure conditions before major impacts on the ecosystem. Autonomic verification, that is, verification of the smart system and ecosystem behavior while emerging and evolving over time, will enhance the early approaches of field testing and test bots, by merging field testing and dynamic analysis approaches with models of human and social behavior to verify the evolving behavior of the systems.

#### *3.2.3 Roadmap.*

- **What is *correctness* for a smart, human-centric ecosystem?** We need to work out new ways to characterize and formalize the quality of a smart, human-centric ecosystems, in the absence of complete and agreed

specifications of the ecosystem. We also need to do this in the presence of emerging, adaptive, and sometimes contradicting behaviors of the systems in the ecosystem.

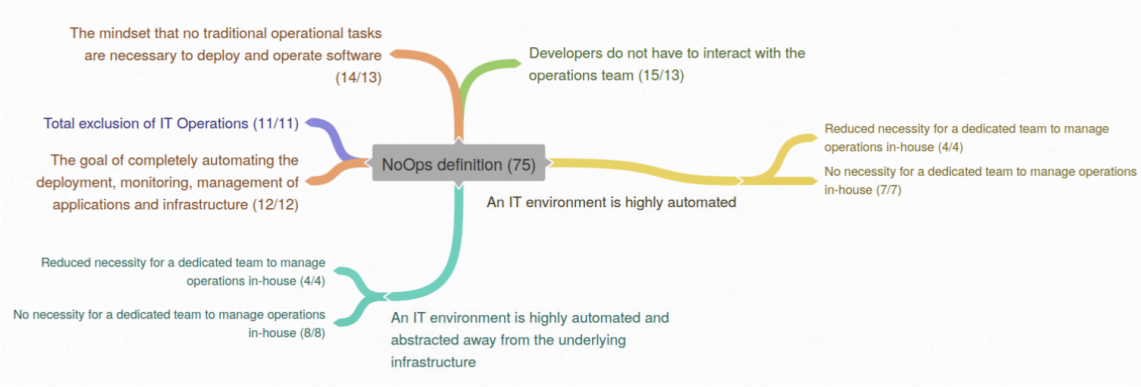
- **How do we handle *unconstrained and implicit interactions* among humans and systems that comprise a smart, human-centric ecosystem?** We need to develop new ways to predict, analyze, and test implicit interactions that emerge and evolve over time. We need studies that investigate better ways to prevent major smart ecosystem failures due to implicit interactions among systems that behave according to their requirements and humans and social groups.
- **How do we model the *behavior of human and social groups* in the presence of evolving and adaptive behavior of smart AI components of human-centric ecosystems?** This requires new studies investigating how to capture behaviors that depend on human personality, gender, emotions, engagement, culture, ethic, age, and more. We need to develop ways to scale from models of humans to models of social groups to assess the quality of large smart, human-centric ecosystems.
- **How do we define *self adaptive strategies*?** We need to carry out studies on how to better handle independently owned, developed and maintained systems within a self adaptive ecosystem. We also need to work our ways to harmonize ecosystem self adaptive strategies with the evolving and adaptive behavior of evolving AI-powered systems within the ecosystem.
- **How do we define self adaptive strategies for smart, human-centric ecosystems?** We need ways to better combine system and human models to include human and social behavior into ecosystem self adaptation strategies. We need to carry out studies on leveraging AI to help to handle independently owned, developed and maintained systems within a self adaptive ecosystem. We also need to combine self adaptive strategies with the evolving and non-deterministic behavior of AI-powered systems within the ecosystem. Ways to combine system and human models to include human and social behavior into ecosystem self adaptation strategies need to be developed and trialled.
- **How do we *predict and mitigate smart ecosystems* failures that arise from implicit interactions of systems that behave correctly, according to their specifications?** We need to identify and harmonize corrective actions among independently-owned and managed systems that behave according to sometimes contradicting requirements.

### 3.3 From NoOps to AIOps

*3.3.1 Trends.* Jabbari et al.'s systematic study [80] spotlights the key recent trends in DevOps: highly technical studies, like Sven's study that focus on the qualities of Infrastructure-as-Code [82], organisational studies that reflects the collaborative dynamics behind DevOps [108], and studies on testing, like Basiris et al.'s drill-like exercises contiguous to Chaos Engineering [16].

The focus on human-centric ecosystems introduces key challenges and complexity of the intrinsically socio-technical relation between the machines and the humans involved as well as the emergent ecology in between. Extreme organisational forms, such as Gualtieri et al.'s No-Operations NoOps [74] and operations managed by individual developers who carry out both Dev and Ops tasks together reduce the organisational space to the bare minimum.

NoOps refers to the objective of developers investing effort only in development activities, with operational activities limited to automatic synchronization, such as *AIOps*, and AI-driven operations [123]. At the time of writing, NoOps




First number in the brackets: number of paragraphs in the analyzed literature that reflect the dimension.
Second number in the brackets: number of sources reflecting the dimension.

Fig. 7. A Mind map diagram that highlights the main dimensions of the NoOps definition

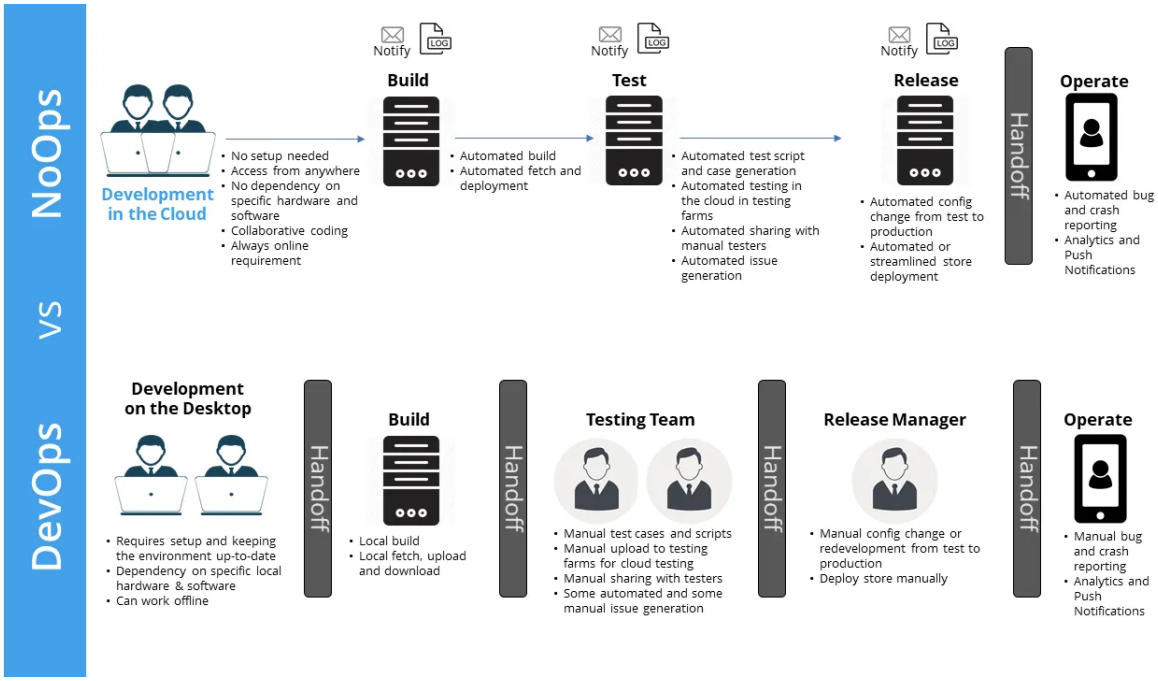
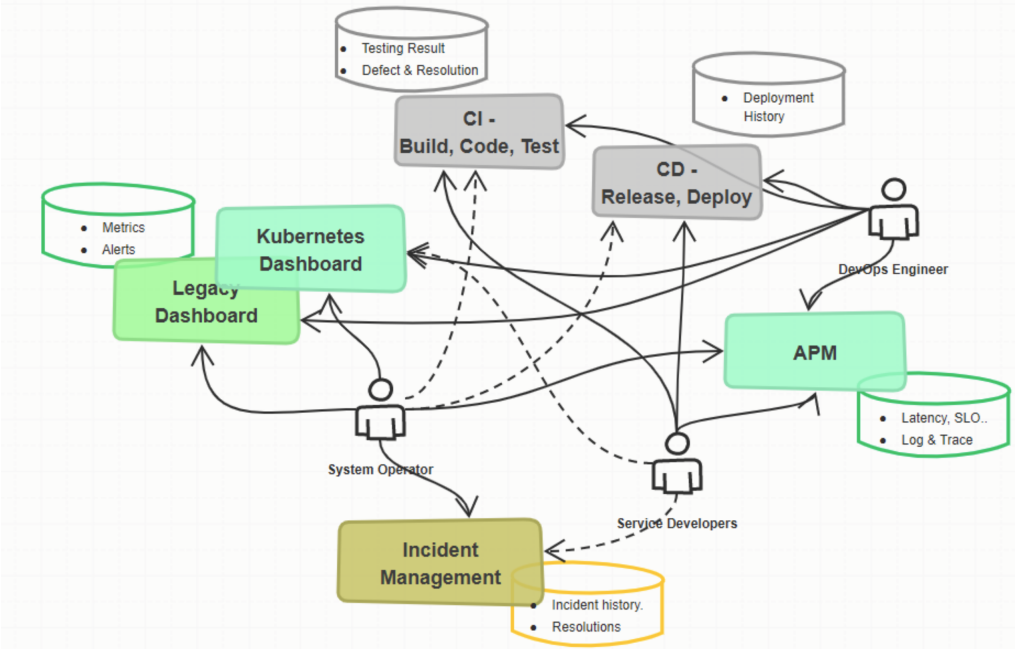



Fig. 8. A comparison of major DevOps and NoOps practices together; synergies and breaking-points are evident

is still a vague concept with some operational definitions that can be drawn from the literature [123], like the visual layout in Figure 7, tailored from [177][6].

[6]https://www.google.com/url?sa=t&source=web&rct=j&opi=89978449&url=https://research.tue.nl/files/199495365/Serban_D.pdf&ved=2ahUKEwi46rD11f2FAxWt_rsIHbArCgEQFnoECBEQAQ&usg=AOvVaw3EV6SyNqJq4BEwxlRTL2Om




Straight arrows indicate data-flows.
Dotted arrows indicate responsibility allocation.
The Artificial Product Manager (APM) stands as the concentrator of infrastructure intelligence and automated operations management.

Fig. 9. AIOps in action, an overview of industrial solutions today

Figure 8, inspired from an exercise originally carried out by Correa[7], illustrates the consequences of highly-individualistic organisational structures on the human-centric ecosystems, by comparing NoOps to classical DevOps structures. We can grasp two striking differences: (i) *automation* that inevitably reduces the role of human-driven operations and changes the relation to operations by developers, and (ii) *AI elements* that simplify the tasks of developers. Reducing human-driven operations forces the process to rely on automation for establishing and maintaining service continuity, while the complexity of human-centric ecosystems requires larger-scale and hybrid computing infrastructures [114], with main challenges related to (i) the synergy that federated AI infrastructure governance mechanisms shall establish and maintain, and (ii) the interactions among humans and AI.

Artificial Intelligence for Operations, AIOps [123], leverages a blend of artificial intelligence, machine learning, and data analytics to automate and enhance the software operational processes. AIOps aims to simplify and accelerate the collection, analysis, and management of data across the various IT strata, to (i) improve operational efficiency, (ii) predict and prevent potential issues, and (iii) quickly deliver accurate services, with a complex intertwining of phases and steps. Figure 9—tailored from Jerry Lee[8]— well exemplifies the layers of complexity of even regular configurations of the AIOps phases and steps.

[7]https://medium.com/@vinuscorrea/noops-the-next-evolution-in-it-operations-or-just-another-stop-on-the-devops-journey-44dab6fb273d
[8]https://tfir.io/aiops-a-long-way-to-get-there/

*3.3.2 Open Challenges.* Contextualizing the complexities of the regular configurations of the AIOps phases and steps in large human-centric ecosystems raises new questions: (a) How can we tackle the emerging synergies among organisations, humans and software ecosystems? (b) How can AI strengthen and cater to those synergies? (c) How can we handle the impact of increasing dynamic machines and infrastructures, like the cloud continuum on operations in human-centric ecosystems? (d) How will humans relate to dynamically evolving ecosystems: Will a *PsyOps* discipline emerge in the software life-cycles, to track the psychological well-being and fatigue-tracking of software developers and operators involved in complex and AI-synergistic software operations?

The forthcoming opportunistic blend of AIOps configurations into human-centric ecosystems, in their highly-individualistic forms, raises several challenges:

- **How can humans handle the increasing data complexity and volume?** We need new ways to manage and analyze vast amounts of data, as well as ensure the quality and consistency of data across multiple sources in complex IT environments.
- **How can we integrate humans into new AIOps Systems?** We need to seamlessly integrate AIOps and humans with the software solutions and IT management tools and processes.
- **How do we maintain accurate AI Models?** We need to develop and maintain accurate AI models that can adapt to changing IT environments. We need continuous learning and model tuning to keep up with new patterns and anomalies.
- **How can we enforce security and privacy by design?** We need to ensure data security and privacy compliance of AIOps platforms that access and process large volumes of sensitive data, while including humans in the loop.
- **How can we close the human–AI Skill gap?** We need to reduce the large skill gap in the workforce when it comes to deploying and managing AIOps solutions as well as understanding and catering to the human-AI synergies [49].

Hybrid human ecosystems shall focus on how humans co-exist and co-create value across the multi-diversity of the cyber-physical infrastructures underlying complex ecosystems. For example a smart city that is constituted by smart homes whose technical infrastructure runs on super-connected cyber-physical cloud continua [115] requires synergy between the various layers of the involved infrastructures as well as decentralised artificial and human intelligence operators. The maintenance and evolution of large organizational structure requires new metrics to account for the emerging functional and non-functional properties emerging, for instance, measuring the mutuality of information and knowledge-sharing among human and AI operators, and measuring the mutual observability of operations for both human and AI operators.

The infrastructure marshalling and management responsibilities previously allocated to so-called orchestrators [82] shall now be delegated to constellations of orchestrators, with the increasing necessity of collaborating, harnessing the power of AI to interact, share knowledge, status updates and operational procedures. In the scope of next-genenration human-centric ecosystems, the classical dilemma of "orchestration vs. choreography"[9] waves the so-called meta-orchestrators, that is, machines capable of harnessing multiple software-defined infrastructure languages, technologies and hardware seamlessly while maintaining 100% operational control and observability.

A key question emerges in this complex scenario: *"What face would a meta-orchestrator of the future exhibit?"* Would established coding mechanisms such as reflexiveness [75] be part of the infrastructure programming models of the future? Would polyglot infrastructure programming encourage maintenance and evolution for infrastructures of the

[9]https://temporal.io/blog/to-choreograph-or-orchestrate-your-saga-that-is-the-question

future? What are the coping mechanisms required for humans to blend their existence within the emergent scenarios of large ecosystems? Is there a risk of *AI fatigue* both for systems and human strata? In summary, three grand challenges for the future arise:

**PsyOps:** the need for instrumented psychology software operations to fit humans into a complex human-centric ecosystems' architecture. Hybrid AI-human interactions within the software lifecycle require new coordination models and psychological studies, beyond current metrics (like socio-technical congruence [109]) and *smells* (like community smells [129]). We need models and studies to understand both *human-AI congruence*, that is, the extent of continuity between human operations and AI assistance, and *AI fatigue*, That is, the excessive use of AI and the consequences that this might have onto the stability of the organisational structure as well as the turmoil around the individual humans within it.

**Meta-Orchestration:** Software engineering related to high-quality software systems' operations shall concentrate on two key ingredients: interchange/integration and AI-continuity. The challenge then becomes providing efficient, effective, and non-invasive engineering techniques to aid the construction and evaluation of meta-orchestrators—machines capable of blending multiple operations perspectives and languages together while maintaining operational awareness and capability over all such perspectives—augmented with artificial intelligence which is rigged by-design to operate in full continuity with such meta-orchestrators, perhaps able to continuously learn from orchestration footprints striving to achieve provocative characteristics such as *explainable orchestration*.

**Superconnectedness:** From an operational perspective, software architectures and the underlying infrastructures are more and more reflecting software-defined infrastructure strata which seem to strive towards fully-connected *meshes* of both data and services to a point in which there is little to no distinction between data products and software services [134]. This characteristic reflects an extent of superconnectedness in the underlying infrastructure, namely the architecture opportunity that emerges by linking together directly all source and sink architecture elements thereby allowing an external—human or AI—operator to enter or govern the emerging ecosystem with novel interfaces beyond conventional CRUD-like operations. Emerging compute options such as *data meshes* [67] are already going in this direction. Software engineering might play a more active role in the process of specifying and evaluating such systems in the future.

*3.3.3 Roadmap.* Although the main challenges that we discussed above seem rather self-contained, they are interrelated. For example, superconnectedness properties of a large-scale human-centric ecosystem reflect explicit and implicit requirements for meta-orchestration. While the challenges reflect complexities which we cannot fathom yet, software engineering principles and practices offer a richness of starting points. The following research questions emerge:

- **Can anthropometrics support psychological operations in software engineering?** Anthropometrics measures the human body and its cognitive instances, and provides categorised data that can be used by (software) designers. We need to study the psychological strains in the software process, especially in relation to the continued use of AI and its interplay with software lifecycles.
- **How can AI fatigue be operationalised during AI-driven software engineering?** We need to study the consequences of the interplay that AI reflects upon the software lifecycle and developers' turnover, software process, product, and humans.
- **Are the principles of cloud and edge engineering valuable starting points to approach meta-orchestration research?** We need to define new principles of cloud and edge engineering to instrument future software designs

and their operation in the context of an extreme interplay between humans, (software-defined) infrastructures, AI, and cyber-physical systems strata.

- **Can explainable AI approaches be combined into explainable orchestration?** We need explainable AI approaches within explainable orchestration to expose both the traces of execution of software within the software lifecycle and the involvement of the AI components in the decisions.
- **How do we characterize human layers of meta-orchestration?** We need to define approaches to support human-centric governance schema behind integrated systems with human government.
- **How can we support the superconnectedness exerted in data meshes?** Data mesh engineering is itself an emerging discipline. We need to define rigorous and explainable approaches to instrument and maintain the superconnectedness, and promote data lineage and other data-* issues as center-stage software design issues.
- **How can we measure the cost of disconnectedness?** The negative instances of connectedness implies hidden costs well beyond our current understanding of technical debt. We need to define metrics of disconnectedness and approaches to models and management disconnectedness.

Each of the above open questions offers hints as to which related disciplines might be able to aid the scientific discourse by means of multi-disciplinarity. For example, cognitive psychology and/or cognitive ergonomics research would play very well within the scope of PsyOps research. Following these premises, at least two possible courses of action are conceivable. First, an endogenous perspective over research in the topics above may start looking for the aforementioned baselines, proceeding in finding a way to blend together such baselines synergistically, for example, blending together cloud continuum research with tiny-scale distributed or federated AIOps to achieve a higher understanding over superconnectedness. Second, an exogenous perspective over research in the topics above may look for opportunities in closely and no so closely related disciplines to understand whether any of the aforementioned challenges were ever encountered elsewhere.

## 4 2030 RESEARCH HORIZON

Below we outline the 2030 research horizon as a combination of the main open issues that we have discussed in this paper, and that comprise a general roadmap for engineering software systems in the era of emerging AI-powered systems and tools. Software is engineered by humans, for humans, and with humans. AI dramatically upsets the role of humans as software developers, as members of engineering teams, and as users, who become active elements of the human-centric software ecosystems and citizen software engineers. It challenges research and practice with new critical questions and important crosscutting concerns.

### 4.1 Key Research Questions

1. **What will software development and the software engineering profession look like in 2030 and beyond?**
   AI-powered development of software systems spotlights a new research program of highly evolving development practices, tools and profession. Core research direction include:

1.1. **Productivity** How do we measure developers' and teams' productivity and experience in hybrid human – AI-powered-agent teams across the entire lifecycle? (see Section 2.1)

1.2. **Tools** How do we better leverage software knowledge, domain knowledge, and development history to improve both engineering tools and AI for SE in general? (see Section 2.2)

1.3. **Profession** How will we interact and work effectively with increasingly powerful AI-powered development agents for engineering useful software systems? (see Section 2.3)
1.4. **Education** What will be the future software body of knowledge and curricula to attract and train excellent software engineers in 2030 and beyond? (see Section 2.5)

2. **What will future software engineering teams look like?**
AI-powered agents dramatically impact human-to-human, human-to-agent and agent-to-agent communication, trust and ethic, and redefine development teams:
2.1. **human-human collaboration** How will AI-powered agents impact human-human collaboration and communication? (see Section 2.3)
2.2. **human-agent collaboration** How will AI-powered agents impact human-agent and agent-agent collaboration and communication? (see Section 2.3)
2.3. **developer-stakeholder communication** How will AI-powered agents change developer-stakeholder and developer-developer empathy and barriers? (see Section 2.3)
2.4. **Creativity, trust and Distributed cognition** What theories of human and AI-powered distributed cognition will we need to use to continue to boost creativity and trust? (see Section 2.3)

3. **What future software engineering processes will we see in 2030 and beyond?**
AI dramatically redefines the software engineering process:
3.1. **Psychological operations in software engineering** How will we support psychological operations and manage AI fatigue in software engineering? (see Section 3.3)
3.2. **Orchestration** How will we merge explainable AI approaches with cloud and edge principles into explainable meta orchestration with human-layers? (see Section 3.3)
3.3. **Superconnectedness** How will we measure and manage superconnectedness and disconnectedness in data meshes? (see Section 3.3)

4. **What will human-centric AI-powered software systems look like by 2030 and beyond?**
AI, extended reality, and the internet of things dramatically change the interactions between humans and AI-powered-systems. They also upset assumptions about both systems that seamlessly adapt to each individual, and systems-of-systems that must quickly evolve into human-centric smart ecosystems, where humans are an integral part and not mere users of the ecosystem:
4.1. **Intelligent AUI** What is the nature of future explainable AI-powered systems for effective and fair-by-design interactions with humans? (see Section 3.1)
4.2. **Smart human-centric ecosystems** How will we engineer and verify future smart human-centric ecosystem? (see Section 3.2)
4.3. **Self-adaptive and evolving behavior** How will we define and verify self-adaptive and evolving AI-powered ecosystems? (see Section 3.2)
4.4. **Citizen software engineers** How will we suitably educate citizen software engineers, and what are the kinds of AI-powered tools for citizen software engineers that will be needed? (see Section 2.4)

### 4.2 Crosscutting Concerns

The many emergent AI system challenges share some important crosscutting concerns about theory, transparency, ethics and inclusiveness with an interdisciplinary approach:

1. **Theory** What theories and metrics will we need to revisit? What can we borrow from other disciplines to capture the disruptive impact of AI on software engineering at all levels: development, productivity, experience, creativity, trust, autonomy and evolution, with both humans and AI-powered agents in the process and the ecosystem?
2. **Transparency** How will we suitably explain AI decisions for transparent interfaces, systems, processes, behavior and evolution?
3. **Ethics** Who is responsible for failures in future AI-heavy systems and processes? What are the key ethical software engineering issues introduced by AI-powered software engineering bots? How will we anticipate and prepare for bad actors as AI is adopted in software tools? How to protect human privacy and creativity?
4. **Inclusiveness** How will we enforce fairness-by-design and respect human diversity and needs within AI-powered processes and products?

The new challenges of AI-powered engineering, AI-augmented socio-technical systems, and multifaceted ethical concerns such as fairness, human aspects and technical dilemmas call for an escalation of inter-disciplinary collaborations with management sciences, HCI, behavioural psychology, social sciences social sciences, behavioural sciences, and cognitive psychology.

## ACKNOWLEDGMENTS

Grundy is supported by ARC Laureate Fellowship FL190100035.

## REFERENCES

[1] Ahmad Abdellatif, Khaled Badran, and Emad Shihab. 2020. MSRBot: Using bots to answer questions from software repositories. *Empirical Software Engineering* 25 (2020), 1834–1863.

[2] Silvia Abrahão, Emilio Insfran, Arthur Sluÿters, and Jean Vanderdonckt. 2021. Model-based intelligent user interface adaptation: challenges and future directions. *Software and Systems Modeling* 20, 5 (2021), 1335–1349. https://doi.org/10.1007/s10270-021-00909-7

[3] Bram Adams and Foutse Khomh. 2020. The diversity crisis of software engineering for artificial intelligence. *IEEE Software* 37, 5 (2020), 104–108.

[4] Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fahmideh, Mst Shamima Aktar, and Tommi Mikkonen. 2023. Towards human-bot collaborative software architecting with chatgpt. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*. 279–285.

[5] Deniz Akdur. 2022. Analysis of software engineering skills gap in the industry. *ACM Transactions on Computing Education* 23, 1 (2022), 1–28.

[6] Pierre A. Akiki. 2018. CHAIN: Developing model-driven contextual help for adaptive user interfaces. *Journal of Systems and Software* 135 (2018), 165–190. https://doi.org/10.1016/J.JSS.2017.10.017

[7] Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu. 2014. Adaptive Model-Driven User Interface Development Systems. *ACM Computing Survey* 47, 1 (2014), 9:1–9:33. https://doi.org/10.1145/2597999

[8] Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu. 2016. Engineering Adaptive Model-Driven User Interfaces. *IEEE Transactions on Software Engineering* 42, 12 (2016), 1118–1147. https://doi.org/10.1109/TSE.2016.2553035

[9] Saleema Amershi, Daniel S. Weld, Mihaela Vorvoreanu, Adam Fourney, Besmira Nushi, Penny Collisson, Jina Suh, Shamsi T. Iqbal, Paul N. Bennett, Kori Inkpen, Jaime Teevan, Ruth Kikin-Gil, and Eric Horvitz. 2019. Guidelines for Human-AI Interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, Glasgow, Scotland, UK, May 04-09, 2019*, Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos (Eds.). ACM, 3. https://doi.org/10.1145/3290605.3300233

[10] Andrew Anderson, Jonathan Dodge, Amrita Sadarangani, Zoe Juozapaitis, Evan Newman, Jed Irvine, Souti Chattopadhyay, Matthew L. Olson, Alan Fern, and Margaret Burnett. 2020. Mental Models of Mere Mortals with Explanations of Reinforcement Learning. *ACM Transactions Interactive Intelligent Systems* 10, 2 (2020), 15:1–15:37. https://doi.org/10.1145/3366485

[11] Sumit Asthana, Hitesh Sajnani, Elena Voyloshnikova, Birendra Acharya, and Kim Herzig. 2023. A Case Study of Developer Bots: Motivations, Perceptions, and Challenges. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1268–1280.

[12] Assaf Avishahar-Zeira and David H Lorenz. 2023. Could No-Code Be Code? Toward a No-Code Programming Language for Citizen Developers. In *Proceedings of the 2023 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. 103–119.

[13] Max Baarspul. 1990. A review of flight simulation techniques. *Progress in aerospace Sciences* 27, 1 (1990), 1–120.

[14] Gagan Bansal, Besmira Nushi, Ece Kamar, Daniel S Weld, Walter S Lasecki, and Eric Horvitz. 2019. Updates in human-ai teams: Understanding and addressing the performance/compatibility tradeoff. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 2429–2437.

[15] Victor R. Basili and Barry T. Perricone. 1984. The influence of organizational factors on software quality and productivity. *Proceedings of the 1st International Conference on Software Engineering* (1984), 154–162.

[16] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. 2016. Chaos Engineering. *IEEE Software* 33, 3 (2016), 35–41. http://dblp.uni-trier.de/db/journals/software/software33.html#BasiriBRHKRR16

[17] Kent Beck. 1999. *Extreme Programming Explained: Embrace Change.* Addison-Wesley Professional, Reading, MA.

[18] Sarah Beecham, Helen Sharp, Tracy Hall, Nathan Baddoo, and Hugh Robinson. 2008. What Do We Know about Developer Motivation? *IEEE Software* (2008). https://doi.org/10.1109/MS.2008.105

[19] Antonia Bertolino, Pietro Braione, Guglielmo De Angelis, Luca Gazzola, Fitsum Meshesha Kifetew, Leonardo Mariani, Matteo Orrú, Mauro Pezzè, Roberto Pietrantuono, Stefano Russo, and Paolo Tonella. 2022. A Survey of Field-based Testing Techniques. *Comput. Surveys* 54, 5 (2022), 92:1–92:39. https://doi.org/10.1145/3447240

[20] Björn Binzer and Till J Winkler. 2022. Democratizing software development: a systematic multivocal literature review and research agenda on citizen development. In *International Conference on Software Business*. Springer, 244–259.

[21] Gordon S. Blair, Nelly Bencomo, and Robert B. France. 2009. Models@ run.time. *Computer* 42, 10 (2009), 22–27. https://doi.org/10.1109/MC.2009.326

[22] Pierre Bourque, Richard E. Fairley, and IEEE Computer Society. 2014. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0* (3rd ed.). IEEE Computer Society Press, Washington, DC, USA.

[23] Frederick P. Brooks. 1995. *The Mythical Man-Month: Essays on Software Engineering* (anniversary ed.). Addison-Wesley Professional.

[24] Margaret M. Burnett, Simone Stumpf, Jamie Macbeth, Stephann Makri, Laura Beckwith, Irwin Kwan, Anicia Peters, and Will Jernigan. 2016. GenderMag: A Method for Evaluating Software's Gender Inclusiveness. *Interactive Computing* 28, 6 (2016), 760–787. https://doi.org/10.1093/IWC/IWV046

[25] Jordi Cabot and Robert Clarisó. 2022. Low code for smart software development. *IEEE Software* 40, 1 (2022), 89–93.

[26] Luiz Fernando Capretz, Pradeep Waychal, Jingdong Jia, Daniel Varona, and Yadira Lizama. 2021. International comparative studies on the software testing profession. *IT Professional* 23, 5 (2021), 56–61.

[27] Edgar Ceh-Varela, Carlos Canto-Bonilla, and Dhimitraq Duni. 2023. Application of Project-Based Learning to a Software Engineering course in a hybrid class environment. *Information and Software Technology* 158 (2023), 107189.

[28] Alexander Chatzigeorgiou, Iftekar Ahmed, Haipeng Cai, Mauro Pezzè, and Denys Poshyvanyk. 2024. Artificial Intelligence for Software Engineering: The Journey so far and the Road ahead. *ACM Transactions on Software Engineering and Methodology* 34, 9 (2024).

[29] Rudrajit Choudhuri, Dylan Liu, Igor Steinmacher, Marco Gerosa, and Anita Sarma. 2024. How Far Are We? The Triumphs and Trials of Generative AI in Learning Software Engineering. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[30] Orges Cico, Letizia Jaccheri, Anh Nguyen-Duc, and He Zhang. 2021. Exploring the intersection between software industry and Software Engineering education-A systematic mapping of Software Engineering Trends. *Journal of Systems and Software* 172 (2021), 110736.

[31] Paul Clements and Linda Northrop. 2002. *Software product lines.* Addison-Wesley Boston.

[32] Hoa Khanh Dam, Truyen Tran, John Grundy, Aditya Ghose, and Yasutaka Kamei. 2019. Towards effective AI-powered agile project management. In *2019 IEEE/ACM 41st international conference on software engineering: new ideas and emerging results (ICSE-NIER)*. IEEE, 41–44.

[33] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley R. Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ronald J. Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek, Serge Mankovski, Raffaela Mirandola, John Mylopoulos, Oscar Nierstrasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Richard D. Schlichting, Dennis B. Smith, João Pedro Sousa, Ladan Tahvildari, Kenny Wong, and Jochen Wuttke. 2010. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers (Lecture Notes in Computer Science, Vol. 7475)*, Rogério de Lemos, Holger Giese, Hausi A. Müller, and Mary Shaw (Eds.). Springer, 1–32. https://doi.org/10.1007/978-3-642-35813-5_1

[34] Ronnie de Souza Santos, William Das Neves Grillo, Djafran Cabral, Catarina De Castro, Nicole Albuquerque, and Cesar França. 2024. Post-Pandemic Hybrid Work in Software Companies: Findings from an Industrial Case Study. In *Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software Engineering*. 68–78.

[35] Ronnie E de Souza Santos and Paul Ralph. 2022. A grounded theory of coordination in remote-first and hybrid software teams. In *Proceedings of the 44th International Conference on Software Engineering*. 25–35.

[36] Tom DeMarco and Timothy Lister. 1999. *Peopleware: Productive Projects and Teams* (2nd ed.). Dorset House Publishing Co., New York.

[37] Paul Denny, Viraj Kumar, and Nasser Giacaman. 2023. Conversing with copilot: Exploring prompt engineering for solving cs1 problems using natural language. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. 1136–1142.

[38] Peter Totterdell Dermot Browne and Mike Norman. 1990. *"Adaptive User Interfaces: Principles"*. Elsevier, United States.

[39] Tilman Deuschel and Ted Scully. 2016. On the Importance of Spatial Perception for the Design of Adaptive User Interfaces. In *10th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO 2016, Augsburg, Germany, September 12-16, 2016*, Giacomo Cabri, Gauthier Picard, and Niranjan Suri (Eds.). IEEE Computer Society, 70–79. https://doi.org/10.1109/SASO.2016.13

[40] Mateusz Dolata and Kevin Crowston. 2023. Making sense of AI systems development. *IEEE Transactions on Software Engineering* (2023).

[41] Schahram Dustdar, Stefan Nastic, and Ognjen Scekic. 2018. *Smart Cities: The Internet of Things, People and Systems* (1st ed.). Springer Publishing Company, Incorporated.

[42] Noura El Moussa, Davide Molinelli, Mauro Pezzè, and Martin Tappler. 2021. Health of smart ecosystems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 1491–1494. https://doi.org/10.1145/3468264.3473137

[43] Omar Elazhary, Margaret-Anne Storey, Neil A. Ernst, and Elise Paradis. 2021. ADEPT: a socio-technical theory of continuous integration. In *Proceedings of the 43rd International Conference on Software Engineering: New Ideas and Emerging Results* (Virtual Event, Spain) *(ICSE-NIER '21)*. IEEE Press, 26–30. https://doi.org/10.1109/ICSE-NIER52604.2021.00014

[44] Anthony Elliott, Brian Peiris, and Chris Parnin. 2015. Virtual reality in software engineering: Affordances, applications, and challenges. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 2. IEEE, 547–550.

[45] Linda Erlenhov, Francisco Gomes de Oliveira Neto, Riccardo Scandariato, and Philipp Leitner. 2019. Current and future bots in software development. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. IEEE, 7–11.

[46] Linda Erlenhov, Francisco Gomes de Oliveira Neto, Riccardo Scandariato, and Philipp Leitner. 2019. Current and Future Bots in Software Development. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 7–11. https://doi.org/10.1109/BotSE.2019.00009

[47] Farjam Eshraghian, Najmeh Hafezieh, Farveh Farivar, and Sergio de Cesare. 2024. AI in software programming: understanding emotional responses to GitHub Copilot. *Information Technology & People* (2024).

[48] Motahhare Eslami, Kristen Vaccaro, Min Kyung Lee, Amit Elazari Bar On, Eric Gilbert, and Karrie Karahalios. 2019. User Attitudes towards Algorithmic Opacity and Transparency in Online Reviewing Platforms. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, CHI 2019, Glasgow, Scotland, UK, May 04-09, 2019*, Stephen A. Brewster, Geraldine Fitzpatrick, Anna L. Cox, and Vassilis Kostakos (Eds.). ACM, 494. https://doi.org/10.1145/3290605.3300724

[49] Tobias Benjamin Fahse and Anuschka Schmitt. 2023. Exploring the Synergies in Human-AI Hybrids: A Longitudinal Analysis in Sales Forecasting. In *American Conference on Information Systems*, Paul A. Pavlou, Vishal Midha, Animesh Animesh, Traci A. Carte, Alexandre R. Graeml, and Alanah Mitchell (Eds.). Association for Information Systems. http://dblp.uni-trier.de/db/conf/amcis/amcis2023.html#FahseS23

[50] Robert Feldt, Francisco G de Oliveira Neto, and Richard Torkar. 2018. Ways of applying artificial intelligence in software engineering. In *Proceedings of the 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. 35–41.

[51] Nicole Forsgren, Jez Humble, and Gene Kim. 2018. *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*. IT Revolution Press, Portland, OR.

[52] Nicole Forsgren, Margaret-Anne Storey, Chandra Maddila, Tom Zimmermann, Brian Houck, and Jenna Butler. 2021. The SPACE of Developer Productivity: There's more to it than you think. *Commun. ACM* 64, 6 (2021), 20–48. https://doi.org/10.1145/3454122.3454124

[53] Eduard Frankford, Clemens Sauerwein, Patrick Bassner, Stephan Krusche, and Ruth Breu. 2024. AI-Tutoring in Software Engineering Education. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training*. 309–319.

[54] Thordur Vikingur Fridgeirsson, Helgi Thor Ingason, Haukur Ingi Jonasson, and Hildur Jonsdottir. 2021. An authoritative study on the near future effect of artificial intelligence on project management knowledge areas. *Sustainability* 13, 4 (2021), 2345.

[55] Ruti Gafni, Itzhak Aviv, Boris Kantsepolsky, Sofia Sherman, Havana Rika, Yariv Itzkovich, and Artem Barger. 2024. Objectivity by design: The impact of AI-driven approach on employees' soft skills evaluation. *Information and Software Technology* 170 (2024), 107430.

[56] Krzysztof Z. Gajos and Krysta Chauncey. 2017. The Influence of Personality Traits and Cognitive Load on the Use of Adaptive User Interfaces. In *Proceedings of the 22nd International Conference on Intelligent User Interfaces, IUI 2017, Limassol, Cyprus, March 13-16, 2017*, George A. Papadopoulos, Tsvi Kuflik, Fang Chen, Carlos Duarte, and Wai-Tat Fu (Eds.). ACM, 301–306. https://doi.org/10.1145/3025171.3025192

[57] Krzysztof Z. Gajos, Katherine Everitt, Desney S. Tan, Mary Czerwinski, and Daniel S. Weld. 2008. Predictability and accuracy in adaptive user interfaces. In *Proceedings of the 2008 Conference on Human Factors in Computing Systems, CHI, Florence, Italy, April 5-10, 2008*, Mary Czerwinski, Arnold M. Lund, and Desney S. Tan (Eds.). ACM, 1271–1274. https://doi.org/10.1145/1357054.1357252

[58] Emily Arteaga Garcia, João Felipe Pimentel, Zixuan Feng, Marco Gerosa, Igor Steinmacher, and Anita Sarma. 2024. How to support ml end-user programmers through a conversational agent. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE)*. IEEE, 629–640.

[59] David Garlan, Shang-Wen Cheng, and Bradley Schmerl. 2003. Increasing System Dependability through Architecture-Based Self-Repair. In *Architecting Dependable Systems*, Rogério de Lemos, Cristina Gacek, and Alexander Romanovsky (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 61–89.

[60] Vahid Garousi, Gorkem Giray, and Eray Tuzun. 2019. Understanding the knowledge gaps of software engineers: An empirical analysis based on SWEBOK. *ACM Transactions on Computing Education (TOCE)* 20, 1 (2019), 1–33.

[61] Daniel Gaspar-Figueiredo, Silvia Abrahão, Marta Fernández-Diego, and Emilio Insfrán. 2023. A Comparative Study on Reward Models for UI Adaptation with Reinforcement Learning. *CoRR* abs/2308.13937 (2023). https://doi.org/10.48550/ARXIV.2308.13937 arXiv:2308.13937

[62] Daniel Gaspar-Figueiredo, Silvia Abrahão, Emilio Insfrán, and Jean Vanderdonckt. 2023. Measuring User Experience of Adaptive User Interfaces using EEG: A Replication Study. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering, EASE 2023, Oulu, Finland, June 14-16, 2023*. ACM, 52–61. https://doi.org/10.1145/3593434.3593452

[63] Daniel Gaspar-Figueiredo, Marta Fernández-Diego, Ruben Nuredini, Silvia Abrahão, and Emilio Insfrán. 2024. Reinforcement Learning-Based Framework for the Intelligent Adaptation of User Interfaces. In *Proceedings of the 16th ACM SIGCHI Symposium on Engineering Interactive Computing*

*Systems, EICS Companion 2024, Cagliari, Italy, June 24-28, 2024*, Michael Nebeling, Lucio Davide Spano, and José Creissac Campos (Eds.). ACM, 40–48. https://doi.org/10.1145/3660515.3661329

[64] Christoph Gebhardt, Brian Hecox, Bas van Opheusden, Daniel Wigdor, James Hillis, Otmar Hilliges, and Hrvoje Benko. 2019. Learning Cooperative Personalized Policies from Gaze Data. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology, UIST 2019, New Orleans, LA, USA, October 20-23, 2019*, François Guimbretière, Michael S. Bernstein, and Katharina Reinecke (Eds.). ACM, 197–208. https://doi.org/10.1145/3332165.3347933

[65] Audrey Girouard, Erin Treacy Solovey, Leanne M. Hirshfield, Evan M. Peck, Krysta Chauncey, Angelo Sassaroli, Sergio Fantini, and Robert J. K. Jacob. 2010. From Brain Signals to Adaptive Interfaces: Using fNIRS in HCI. In *Brain-Computer Interfaces - Applying our Minds to Human-Computer Interaction*, Desney S. Tan and Anton Nijholt (Eds.). Springer, 221–237. https://doi.org/10.1007/978-1-84996-272-8_13

[66] Camille Gobert, Kashyap Todi, Gilles Bailly, and Antti Oulasvirta. 2019. SAM: a modular framework for self-adapting web menus. In *Proceedings of the 24th International Conference on Intelligent User Interfaces, IUI 2019, Marina del Ray, CA, USA, March 17-20, 2019*, Wai-Tat Fu, Shimei Pan, Oliver Brdiczka, Polo Chau, and Gaelle Calvary (Eds.). ACM, 481–484. https://doi.org/10.1145/3301275.3302314

[67] Abel Goedegebuure, Indika Kumara, Stefan Driessen, Dario Di Nucci, Geert Monsieur, Willem jan van den Heuvel, and Damian Andrew Tamburri. 2024. Data Mesh: a Systematic Gray Literature Review. arXiv:2304.01062

[68] Valentina Golendukhina, Valentina Lenarduzzi, and Michael Felderer. 2022. What is software quality for AI engineers? Towards a thinning of the fog. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI*. 1–9.

[69] Don Gotterbarn, Keith Miller, and Simon Rogerson. 1997. Software engineering code of ethics. *Commun. ACM* 40, 11 (1997), 110–118.

[70] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. 2018. What happens when software developers are (un)happy. *Journal of Systems and Software* 140 (2018), 32–47. https://doi.org/10.1016/j.jss.2018.02.041

[71] Shalom Greene, Himanshu Thapliyal, and Allison Caban-Holt. 2016. A Survey of Affective Computing for Stress Detection: Evaluating technologies in stress detection for better health. *IEEE Consumer Electronics Magazine* 5, 4 (2016), 44–56. https://doi.org/10.1109/MCE.2016.2590178

[72] Michaela Greiler, Margaret-Anne Storey, and Abi Noda. 2023. An Actionable Framework for Understanding and Improving Developer Experience. *IEEE Transactions on Software Engineering* 49, 4 (2023), 1411–1425. https://doi.org/10.1109/TSE.2023.1234567

[73] John C. Grundy. 2020. Human-centric Software Engineering for Next Generation Cloud- and Edge-based Smart Living Applications. In *20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGRID 2020, Melbourne, Australia, May 11-14, 2020*. IEEE, 1–10. https://doi.org/10.1109/CCGRID49817.2020.00-93

[74] Mike Gualtieri, Sandy Carielli, and Cheryl McKinnon. 2011. I Don't Want DevOps. I Want NoOps. https://go.forrester.com/blogs/11-02-07-i_dont_want_devops_i_want_noops/.

[75] Kris Gybels, Roel Wuyts, Stéphane Ducasse, and Maja D'Hondt. 2006. Inter-language reflection: A conceptual model and its implementation. *Computer Languages, Systems & Structures* 32, 2-3 (2006), 109 – 124. https://doi.org/10.1016/j.cl.2005.10.003

[76] Khadija Hanifi, Orcun Cetin, and Cemal Yilmaz. 2023. On ChatGPT: perspectives from software engineering students. In *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*. IEEE, 196–205.

[77] Catherine Hicks, Carol Lee, and Kristen Foster-Marks. 2024. The New Developer: AI Skill Threat, Identity Change; Developer Thriving in the Transition to AI-Assisted Software Development. https://doi.org/10.31234/osf.io/2gej5

[78] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. arXiv:2308.10620 [cs.SE] https://arxiv.org/abs/2308.10620

[79] Andrew Hunt and David Thomas. 1999. *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional, Boston, MA.

[80] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. 2016. What is DevOps? A Systematic Mapping Study on Definitions and Practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016* (Edinburgh, Scotland, UK) *(XP '16 Workshops)*. Association for Computing Machinery, New York, NY, USA, Article 12. https://doi.org/10.1145/2962695.2962707

[81] Victoria Jackson, Bogdan Vasilescu, Daniel Russo, Paul Ralph, Rafael Prikladnicki, Maliheh Izadi, Sarah D'Angelo, Sarah Inman, Anielle Lisboa, and André van der Hoek. 2025. The Impact of Generative AI on Creativity in Software Development: A Research Agenda. *ACM Transaction on Software Engineering and Methodology* 34, 4 (April 2025).

[82] Sven Johann. 2017. Kief Morris on Infrastructure as Code. *IEEE Software* 34, 1 (2017), 117–120. http://dblp.uni-trier.de/db/journals/software/software34.html#Johann17

[83] Capers Jones. 1986. Measuring programmer productivity. *IBM Systems Journal* 22, 1 (1986), 36–48.

[84] Eirini Kalliamvakou, Christian Bird, Thomas Zimmermann, Andrew Begel, Robert DeLine, and Daniel M German. 2017. What makes a great manager of software engineers? *IEEE Transactions on Software Engineering* 45, 1 (2017), 87–106.

[85] Eirini Kalliamvakou, Sida Peng, Peter Cihon, and Mert Demirer. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. *Commun. ACM* 66, 6 (2023), 28–37. https://doi.org/10.1145/3454122.3454124

[86] Christian Kästner and Eunsuk Kang. 2020. Teaching software engineering for AI-enabled systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering Education and Training* (Seoul, South Korea) *(ICSE-SEET '20)*. Association for Computing Machinery, New York, NY, USA, 45–48. https://doi.org/10.1145/3377814.3381714

[87] Brian W. Kernighan and Dennis M. Ritchie. 1988. *The C Programming Language* (2nd ed.). Prentice Hall, Englewood Cliffs, NJ.

[88] Dron Khanna, Emily Laue Christensen, Saagarika Gosu, Xiaofeng Wang, and Maria Paasivaara. 2024. Hybrid Work meets Agile Software Development: A Systematic Mapping Study. In *Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects*

*of Software Engineering*. 57–67.

[89] Vassilka D Kirova, Cyril S Ku, Joseph R Laracy, and Thomas J Marlowe. 2024. Software engineering education must adapt and evolve for an llm environment. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1*. 666–672.

[90] Antti Knutas, Victoria Palacin, Giovanni Maccani, and Markus Helfert. 2019. Software engineering in civic tech a case study about code for ireland. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. IEEE, 41–50.

[91] Kathrin Komp-Leukkunen. 2024. How ChatGPT shapes the future labour market situation of software engineers: A Finnish Delphi study. *Futures* 160 (2024), 103382.

[92] Mohammad Amin Kuhail, Sujith Samuel Mathew, Ashraf Khalil, Jose Berengueres, and Syed Jawad Hussain Shah. 2024. "Will I be replaced?" Assessing ChatGPT's effect on software development and programmer perceptions of AI tools. *Science of Computer Programming* 235 (2024), 103111.

[93] Todd Kulesza, Margaret M. Burnett, Weng-Keen Wong, and Simone Stumpf. 2015. Principles of Explanatory Debugging to Personalize Interactive Machine Learning. In *Proceedings of the 20th International Conference on Intelligent User Interfaces, IUI 2015, Atlanta, GA, USA, March 29 - April 01, 2015*, Oliver Brdiczka, Polo Chau, Giuseppe Carenini, Shimei Pan, and Per Ola Kristensson (Eds.). ACM, 126–137. https://doi.org/10.1145/2678025.2701399

[94] Kati Kuusinen, Helen Petrie, Fabian Fagerholm, and Tommi Mikkonen. 2016. Flow, Intrinsic Motivation, and Developer Experience in Software Engineering. In *Agile Processes, in Software Engineering, and Extreme Programming*, Helen Sharp and Tracy Hall (Eds.). Springer International Publishing, Cham, 104–117.

[95] Samuli Laato, Matti Mäntymäki, AKM Najmul Islam, Sami Hyrynsalmi, and Teemu Birkstedt. 2023. Trends and Trajectories in the Software Industry: implications for the future of work. *Information Systems Frontiers* 25, 2 (2023), 929–944.

[96] Samuli Laato, Miika Tiainen, AKM Najmul Islam, and Matti Mäntymäki. 2022. How to explain AI systems to end users: a systematic literature review and research agenda. *Internet Research* 32, 7 (2022), 1–31.

[97] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N Nguyen. 2017. Bug localization with combination of deep learning and information retrieval. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. IEEE, 218–229.

[98] Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. 2021. A systematic review of API evolution literature. *ACM Computing Surveys (CSUR)* 54, 8 (2021), 1–36.

[99] Thomas Langerak, Sammy Joe Christen, Mert Albaba, Christoph Gebhardt, and Otmar Hilliges. 2022. MARLUI: Multi-Agent Reinforcement Learning for Goal-Agnostic Adaptive UIs. *CoRR* abs/2209.12660 (2022). https://doi.org/10.48550/ARXIV.2209.12660 arXiv:2209.12660

[100] Filippo Lanubile, Christof Ebert, Rafael Prikladnicki, and Aurora Vizcaíno. 2010. Collaboration tools for global software engineering. *IEEE software* 27, 2 (2010), 52.

[101] Talia Lavie and Joachim Meyer. 2010. Benefits and costs of adaptive user interfaces. *International Journal of Human-Computer Studies* 68, 8 (2010), 508 – 524. https://doi.org/10.1016/j.ijhcs.2010.01.004

[102] Paul Luo Li, Amy J Ko, and Jiamin Zhu. 2015. What makes a great software engineer?. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 700–710.

[103] Tianyi Li, Mihaela Vorvoreanu, Derek DeBellis, and Saleema Amershi. 2023. Assessing Human-AI Interaction Early through Factorial Surveys: A Study on the Guidelines for Human-AI Interaction. *ACM Transactions Computer Human Interactions* 30, 5 (2023), 69:1–69:45. https://doi.org/10.1145/3511605

[104] David Lo. 2023. Trustworthy and Synergistic Artificial Intelligence for Software Engineering: Vision and Roadmaps. In *International Conference on Software Engineering: Future of Software Engineering*. IEEE.

[105] Qinghua Lu, Liming Zhu, Xiwei Xu, Jon Whittle, David Douglas, and Conrad Sanderson. 2022. Software engineering for responsible AI: An empirical study and operationalised patterns. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice*. 241–242.

[106] Qinghua Lu, Liming Zhu, Xiwei Xu, Jon Whittle, Didar Zowghi, and Aurelie Jacquet. 2023. Responsible ai pattern catalogue: A collection of best practices for ai governance and engineering. *Comput. Surveys* (2023).

[107] Silverio Martínez-Fernández, Justus Bogner, Xavier Franch, Marc Oriol, Julien Siebert, Adam Trendowicz, Anna Maria Vollmer, and Stefan Wagner. 2022. Software engineering for AI-based systems: a survey. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 2 (2022), 1–59.

[108] T. Masombuka and Ernest Mnkandla. 2018. A DevOps collaboration culture acceptance model.. In *SAICSIT*, Sue Petratos, Johan Van Niekerk, and Bertram Haskins (Eds.). ACM, 279–285. http://dblp.uni-trier.de/db/conf/saicsit/saicsit2018.html#MasombukaM18

[109] Wolfgang Mauerer, Mitchell Joblin, Damian A. Tamburri, Carlos Paradis, Rick Kazman, and Sven Apel. 2022. In Search of Socio-Technical Congruence: A Large-Scale Longitudinal Study. *IEEE Transactions on Software Engineering* 48, 8 (2022), 3159–3184. https://doi.org/10.1109/TSE.2021.3082074

[110] Marshall McLuhan. 1977. LAWS OF THE MEDIA. *ETC: A Review of General Semantics* 34, 2 (1977), 173–179. http://www.jstor.org/stable/42575246

[111] Leonel Merino, Mircea Lungu, and Christoph Seidl. 2020. Unleashing the potentials of immersive augmented reality for software engineering. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 517–521.

[112] Marjan Mernik, Jan Heering, and Anthony M Sloane. 2005. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)* 37, 4 (2005), 316–344.

[113] Nesrine Mezhoudi and Jean Vanderdonckt. 2021. Toward a Task-driven Intelligent GUI Adaptation by Mixed-initiative. *International Journal on Human Computer Interactions* 37, 5 (2021), 445–458. https://doi.org/10.1080/10447318.2020.1824742

[114] Aleksandar Milenkoski, Alexandru Iosup, Samuel Kounev, Kai Sachs, Piotr Rygielski, Jason Ding, Walfredo Cirne, and Florian Rosenberg. 2013. *Cloud Usage Patterns: A Formalism for Description of Cloud Usage Scenarios.* Technical Report SPEC-RG-2013-001 v.1.0.1. SPEC Research Group - Cloud Working Group, Standard Performance Evaluation Corporation (SPEC), 7001 Heritage Village Plaza Suite 225, Gainesville, VA 20155, USA. http://research.spec.org/fileadmin/user_upload/documents/rg_cloud/endorsed_publications/SPEC-RG-2013-001_CloudUsagePatterns.pdf

[115] Dejan S. Milojicic. 2020. The Edge-to-Cloud Continuum. *IEEE Computer* 53, 11 (2020), 16–25. http://dblp.uni-trier.de/db/journals/computer/computer53.html#Milojicic20a

[116] Ivan Mistrík, John Grundy, Andre Van der Hoek, and Jim Whitehead. 2010. *Collaborative software engineering: challenges and prospects.* Springer.

[117] Hausi A. Müller, Mauro Pezzè, and Mary Shaw. 2008. Visibility of control in adaptive systems. In *Proceedings of the 2nd international workshop on Ultra-large-scale software-intensive systems, ULSSIS@ICSE 2008, Leipzig, Germany, May 10-11, 2008*, Kevin J. Sullivan and Rick Kazman (Eds.). ACM, 23–26. https://doi.org/10.1145/1370700.1370707

[118] Alex Murphy, Ben Kelly, Kai Bergmann, Kyrylo Khaletskyy, Rory V O'Connor, and Paul M Clarke. 2019. Examining unequal gender distribution in software engineering. In *Systems, Software and Services Process Improvement: 26th European Conference, EuroSPI 2019, Edinburgh, UK, September 18–20, 2019, Proceedings 26.* Springer, 659–671.

[119] Peter Naur. 1986. Programming as Theory Building. *Microprocessing and Microprogramming* 15, 5 (1986), 253–261.

[120] Peter Naur and Brian Randell. 1969. *Software Engineering: Report on a Conference Sponsored by the NATO Science Committees.* Technical Report. Scientific Affairs Division, NATO.

[121] A.F. Norcio and J. Stanley. 1989. Adaptive human-computer interfaces: a literature survey and perspective. *IEEE Transactions on Systems, Man, and Cybernetics* 19, 2 (1989), 399–408. https://doi.org/10.1109/21.31042

[122] L. Northrop, Peter Feiler, R.P. Gabriel, John Goodenough, Rick Linger, Thomas Longstaff, Rick Kazman, M. Klein, Douglas Schmidt, Kevin Sullivan, and Kurt Wallnau. 2006. *Ultra-Large-Scale Systems - The Software Challenge of the Future.* Technical Report. Software Engineering Institute – Carnegie Mellon.

[123] Paolo Notaro, Jorge Cardoso, and Michael Gerndt. 2020. A Systematic Mapping Study in AIOps. In *ICSOC Workshops (Lecture Notes in Computer Science, Vol. 12632)*, Hakim Hacid, Fatma Outay, Hye young Paik, Amira Alloum, Marinella Petrocchi, Mohamed Reda Bouadjenek, Amin Beheshti, Xumin Liu, and Abderrahmane Maaradji (Eds.). Springer, 110–123. http://dblp.uni-trier.de/db/conf/icsoc/icsoc2020w.html#NotaroCG20

[124] Nargess Nourbakhsh, Fang Chen, Yang Wang, and Rafael A. Calvo. 2017. Detecting Users' Cognitive Load by Galvanic Skin Response with Affective Interference. *ACM Transactions Interactions on Intelligent Systems* 7, 3 (2017), 12:1–12:20. https://doi.org/10.1145/2960413

[125] John P. Oliver and Srivastava Sanjay. 1999. The Big Five Trait taxonomy: History, measurement, and theoretical perspectives. In *Handbook of personality: Theory and research*, L. A. Pervin and O. P. John (Eds.). Guilford Pres., 102–138.

[126] John P. Oliver and Srivastava Sanjay. 2008. *Perspectives on personality (6th edition).* Pearson.

[127] Marten Oltrogge, Erik Derr, Christian Stransky, Yasemin Acar, Sascha Fahl, Christian Rossow, Giancarlo Pellegrino, Sven Bugiel, and Michael Backes. 2018. The rise of the citizen developer: Assessing the security impact of online app generators. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 634–647.

[128] Ipek Ozkaya. 2022. A paradigm shift in automating software engineering tasks: Bots. *IEEE Software* 39, 5 (2022), 4–8.

[129] Fabio Palomba, Damian Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, and Alexander Serebrenik. 2021. Beyond Technical Aspects: How Do Community Smells Influence the Intensity of Code Smells? *IEEE Transactions on Software Engineering* 47, 1 (2021), 108–129. https://doi.org/10.1109/TSE.2018.2883603

[130] Aastha Pant, Rashina Hoda, Chakkrit Tantithamthavorn, and Burak Turhan. 2024. Ethics in AI through the practitioner's view: a grounded theory literature review. *Empirical Software Engineering* 29, 3 (2024), 67.

[131] Chris Paton, Andre W Kushniruk, Elizabeth M Borycki, Mike English, and Jim Warren. 2021. Improving the Usability and Safety of Digital Health Systems: The Role of Predictive Human-Computer Interaction Modeling. *Journal of Medical Internet Research* 23, 5 (May 2021), e25281.

[132] Sampo V. Paunonen. 2003. Big Five factors of personality and replicated predictions of behavior. *Journal of Personality and Social Psychology* 84, 2 (2003), 411–424. https://doi.org/10.1037/0022-3514.84.2.411

[133] Olga Petrovska, Lee Clift, Faron Moller, and Rebecca Pearsall. 2024. Incorporating Generative AI into Software Development Education. In *Proceedings of the 8th Conference on Computing Education Practice*. 37–40.

[134] Giovanni Quattrocchi, Willem-Jan van den Heuvel, and Damian Andrew Tamburri. 2024. The Data Product-service Composition Frontier: A Hybrid Learning Approach. *ACM Transactions Manage. Inf. Syst.* 15, 1, Article 6 (mar 2024), 22 pages. https://doi.org/10.1145/3649319

[135] Emilee J. Rader, Kelley Cotter, and Janghee Cho. 2018. Explanations as Mechanisms for Supporting Algorithmic Transparency. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, CHI 2018, Montreal, QC, Canada, April 21-26, 2018*, Regan L. Mandryk, Mark Hancock, Mark Perry, and Anna L. Cox (Eds.). ACM, 103. https://doi.org/10.1145/3173574.3173677

[136] Charles Rich and Richard Waters. 1988. The Programmer's Apprentice Project: A Research Overview. *Computer* 21 (12 1988), 10 – 25. https://doi.org/10.1109/2.86782

[137] Tim Rietz. 2019. Designing a conversational requirements elicitation system for end-users. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE, 452–457.

[138] Pierre N Robillard and Martin P Robillard. 2000. Types of collaborative work in software engineering. *Journal of Systems and Software* 53, 3 (2000), 219–224.

[139] Gema Rodríguez-Pérez, Reza Nadri, and Meiyappan Nagappan. 2021. Perceived diversity in software engineering: a systematic literature review. *Empirical Software Engineering* 26 (2021), 1–38.

[140] Babak Darvish Rouhani, Mohd Naz'ri Mahrin, Fatemeh Nikpay, Rodina Binti Ahmad, and Pourya Nikfard. 2015. A systematic literature review on Enterprise Architecture Implementation Methodologies. *information and Software Technology* 62 (2015), 1–20.

[141] Daniel Russo. 2024. Navigating the complexity of generative ai adoption in software engineering. *ACM Transactions on Software Engineering and Methodology* (2024).

[142] Elizabeth B.-N. Sanders and Pieter Jan Stappers. 2008. Co-creation and the new landscapes of design. *CoDesign* 4, 1 (2008), 5–18. https://doi.org/10.1080/15710880701875068 arXiv:http://dx.doi.org/10.1080/15710880701875068

[143] Jaakko Sauvola, Sasu Tarkoma, Mika Klemettinen, Jukka Riekki, and David Doermann. 2024. Future of software development with generative AI. *Automated Software Engineering* 31, 1 (2024), 26.

[144] J. Schneider-Hufschmidt, T. Kühme, and Malinowski U. 1993. *"Adaptive User Interfaces: Principles and Practice"*. North Holland, London.

[145] Concetta Semeraro, Mario Lezoche, Hervé Panetto, and Michele Dassisti. 2021. Digital twin paradigm: A systematic literature review. *Computers in Industry* 130 (2021), 103469.

[146] Rifat Ara Shams, Didar Zowghi, and Muneera Bano. 2023. AI and the quest for diversity and inclusion: a systematic literature review. *AI and Ethics* (2023), 1–28.

[147] Ben Shneiderman. 2020. Bridging the gap between ethics and practice: guidelines for reliable, safe, and trustworthy human-centered AI systems. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 10, 4 (2020), 1–31.

[148] James Shore and Shane Warden. 2007. *The Art of Agile Development*. O'Reilly Media, Sebastopol, CA.

[149] Martin Stancek, Ivan Polasek, Tibor Zalabai, Juraj Vincur, Rodi Jolak, and Michel Chaudron. 2024. Collaborative software design and modeling in virtual reality. *Information and Software Technology* 166 (2024), 107369.

[150] Vladimir Stantchev, Olaf Radant, and Ricardo Colomo-Palacios. 2016. Assessment of continuing educational measures in software engineering: A view from the industry. *International Journal of Engineering Education* 32, 2B (2016), 905–914.

[151] Åsne Stige, Efpraxia D Zamani, Patrick Mikalef, and Yuzhen Zhu. 2023. Artificial intelligence (AI) for user experience (UX) design: a systematic literature review and future research agenda. *Information Technology & People* (2023).

[152] Margaret-Anne Storey, Daniel Russo, Nicole Novielli, Takashi Kobayashi, and Dong Wang. 2024. A Disruptive Research Playbook for Studying Disruptive Innovations. *ArXiv e-prints* (2024).

[153] Margaret-Anne Storey, Christoph Treude, Arie Van Deursen, and Li-Te Cheng. 2010. The impact of social media on software engineering practices and tools. In *Proceedings of the FSE/SDP workshop on Future of software engineering research*. 359–364.

[154] Margaret-Anne Storey, Thomas Zimmermann, Christian Bird, Jacek Czerwonka, Brendan Murphy, and Eirini Kalliamvakou. 2021. Towards a Theory of Software Developer Job Satisfaction and Perceived Productivity. *IEEE Transactions on Software Engineering* 47, 10 (2021), 2125–2142. https://doi.org/10.1109/TSE.2019.2944354

[155] R. Subramanyam and M. S. Krishnan. 2003. Understanding and improving software productivity: an empirical study. *Commun. ACM* 46, 2 (2003), 73–77.

[156] Thomas Süße, Maria Kobert, Simon Grapenthin, and Bernd-Friedrich Voigt. 2023. AI-Powered Chatbots and the Transformation of Work: Findings from a Case Study in Software Development and Software Engineering. In *Working Conference on Virtual Enterprises*. Springer, 689–705.

[157] Stephanie D Teasley, Lisa A Covi, Mayuram S. Krishnan, and Judith S Olson. 2002. Rapid software development through team collocation. *IEEE Transactions on software engineering* 28, 7 (2002), 671–683.

[158] Kashyap Todi, Gilles Bailly, Luis Leiva, and Antti Oulasvirta. 2021. Adapting User Interfaces with Model-Based Reinforcement Learning. In *Proceedings of the 2021 Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 573, 13 pages. https://doi.org/10.1145/3411764.3445497

[159] Kashyap Todi and Tanya R. Jonker. 2023. A Framework for Computational Design and Adaptation of Extended Reality User Interfaces. *CoRR* abs/2309.04025 (2023). https://doi.org/10.48550/ARXIV.2309.04025 arXiv:2309.04025

[160] James E. Trumbly, Kirk P. Arnett, and Peter C. Johnson. 1994. Productivity gains via an adaptive user interface: an empirical analysis. *International Journal of Human-Computer Studies* 40, 1 (1994), 63–81. https://doi.org/10.1006/ijhc.1994.1004

[161] Michele Tufano, Anisha Agarwal, Jinu Jang, Roshanak Zilouchian Moghaddam, and Neel Sundaresan. 2024. AutoDev: Automated AI-Driven Development. arXiv:2403.08299 [cs.SE] https://arxiv.org/abs/2403.08299

[162] Marcel Valovỳ and Alena Buchalcevova. 2023. The Psychological Effects of AI-Assisted Programming on Students and Professionals. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 385–390.

[163] Jean Vanderdonckt, Sara Bouzit, Gaëlle Calvary, and Denis Chêne. 2020. Exploring a Design Space of Graphical Adaptive Menus: Normal vs. Small Screens. *ACM Transactions Interactive Intelligent Systems* 10, 1 (2020), 2:1–2:40. https://doi.org/10.1145/3237190

[164] Stefan Wagner and Melanie Ruhe. 2019. A Systematic Review of Productivity Factors in Software Development. *arXiv preprint arXiv:1801.06475* (2019).

[165] Zhiyuanl Xia Wan, David Lo, and David Lo. [n. d.]. How does machine learning change software development practices?.(2019). *IEEE Transactions on Software Engineering* ([n. d.]), 1–14.

[166] Junjie Wang, Ye Yang, Song Wang, Jun Hu, and Qing Wang. 2022. Context-and fairness-aware in-process crowdworker recommendation. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022), 1–31.

[167] Wei Wang, Hourieh Khalajzadeh, John C. Grundy, Anuradha Madugalla, and Humphrey O. Obie. 2024. Adaptive User Interfaces for Software Supporting Chronic Disease. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society, ICSE-SEIS2024, Lisbon, Portugal, April 14-20, 2024*. ACM, 118–129. https://doi.org/10.1145/3639475.3640104

[168] Yi Wang and David Redmiles. 2019. Implicit gender biases in professional software development: An empirical study. In *2019 IEEE/ACM 41st international conference on software engineering: Software engineering in society (ICSE-SEIS)*. IEEE, 1–10.

[169] Barbara Weber, Thomas Fischer, and René Riedl. 2021. Brain and autonomic nervous system activity measurement in software engineering: A systematic literature review. *Journal of Systems and Software* 178 (2021), 110946. https://doi.org/10.1016/j.jss.2021.110946

[170] Gerald M. Weinberg. 1971. *The Psychology of Computer Programming*. Van Nostrand Reinhold, New York.

[171] Justin D Weisz, Michael Muller, Stephanie Houde, John Richards, Steven I Ross, Fernando Martinez, Mayank Agarwal, and Kartik Talamadupula. 2021. Perfection not required? Human-AI partnerships in code translation. In *26th International Conference on Intelligent User Interfaces*. 402–412.

[172] Jim Whitehead. 2007. Collaboration in software engineering: A roadmap. In *Future of Software Engineering (FOSE'07)*. IEEE, 214–225.

[173] Enes Yigitbas, Ivan Jovanovikj, Kai Biermeier, Stefan Sauer, and Gregor Engels. 2020. Integrated model-driven development of self-adaptive user interfaces. *Softw. Syst. Model.* 19, 5 (2020), 1057–1081. https://doi.org/10.1007/S10270-020-00777-7

[174] Enes Yigitbas, Hagen Stahl, Stefan Sauer, and Gregor Engels. 2017. Self-adaptive UIs: Integrated Model-Driven Development of UIs and Their Adaptations. In *Modelling Foundations and Applications - 13th European Conference, ECMFA@STAF 2017, Marburg, Germany, July 19-20, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10376)*, Anthony Anjorin and Huáscar Espinoza (Eds.). Springer, 126–141. https://doi.org/10.1007/978-3-319-61482-3_8

[175] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024. AutoCodeRover: Autonomous Program Improvement. arXiv:2404.05427 [cs.SE] https://arxiv.org/abs/2404.05427

[176] Lamia Zouhaier, Yosra Ben Dali Hlaoui, and Leila Ben Ayed. 2023. Adaptive user interface based on accessibility context. *Multim. Tools Appl.* 82, 23 (2023), 35621–35650. https://doi.org/10.1007/S11042-023-14390-5

[177] Dragos Mihai Şerban. 2021. *Demystifying NoOps: operational model, challenges and insights from the trenches*. Master's thesis. Eindhoven University of Technology.