ISSN 2186-7437

NII Shonan Meeting Report

No. 191

Human Aspects of Software Engineering

Thomas Fritz (University of Zurich, Switzerland)
Yasutaka Kamei (Kyushu University, Japan)
Thomas Zimmermann (Microsoft, USA)

March 6–9, 2023

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Human Aspects of Software Engineering

Organizers:
Thomas Fritz (University of Zurich, Switzerland)
Yasutaka Kamei (Kyushu University, Japan)
Thomas Zimmermann (Microsoft, USA)

March 6–9, 2023

Report created by André N. Meyer (University of Zurich, Switzerland)

Abstract

Software is built by humans. Software developers are the ones who develop and evolve code, that elicit requirements, test the software, and talk to their teammates to coordinate. Yet, traditionally, research has focused to a large extent on normative processes and artefacts – how developers ought to develop software, the digital objects developers have created or modified, measuring their output, and collecting data from software repositories.

While this focus on ideal work processes and developers' output can provide interesting and relevant insights, it falls short when the goal is to better understand the humans in the process, such as the cognitive demands and emotions they experience, and the individual differences between developers while they create and evolve the output data. Especially since these human aspects can have a significant effect on the output and its quality, the better we understand the human in the process, the better we can support the software development endeavor, and the better software quality we can achieve.

This meeting will bring together leading researchers to discuss current and future trends and challenges related to human aspects in software engineering, for example:

- How to best support the human in the process, especially given the increasing cognitive demands and the interleaving of work and life. The complexity of software is increasing and developers now have to work with complex engineering pipelines.
- How to evaluate and design AI-powered tools from a human perspective. AI-powered code completion tools such as Copilot are revolutionizing software development. We will discuss how to evaluate such systems from a human perspective and how to measure their impact on productivity. We will also discuss what new experiences can be created across the entire development life cycle to support humans in the software process more efficiently.

- How to improve focus and reduce distractions for software developers. The productivity of software developers has received significant attention in industry, for example, at Google, Microsoft, Facebook, and other companies. We plan to discuss how to measure and improve productivity of software engineers.
- How to increase software developer's well-being. While it is common sense that well-being is important for employees, there has been limited research in the context of software engineering. We plan to discuss research opportunities around this topic.
- How to support software developers working remotely and the new future of work. The coronavirus pandemic has significantly disrupted how people work and many employees now work remotely. We plan to discuss challenges that emerge and how we can best support developers in a remote setting. We will also plan to discuss a new future of software development that supports sustainable distributed remote work.
- How to support developers using biometric information. Biometric sensors offer a unique opportunity to collect data about software development and correlate with other data signals to learn more about how developers work and what cognitive demands they are facing. We plan to discuss research opportunities around this topic.
- How to navigate ethics and privacy issues. We plan to discuss common issues around ethics and privacy in empirical research and how to do research responsibility.
- How to educate future researchers in this domain. We will focus on compiling materials for master students who want to start research projects on human aspects in software engineering. In addition, we will focus on compiling review guidelines and best practices for experienced researchers to improve the quality of ongoing research.

Meeting format. The meeting will be highly interactive. It will include a mix of short lightning talks by the attendees, followed by breakout sessions on common topics of interest. We will have extended presentations from industrial participants on how human aspects of software engineering are considered in industry. We will closely involve the participants in the design of the agenda and the definition of the desired meeting outcomes (e.g., collaborations, books, publications, special issues, follow-up meetings, etc.)

Academic impact. We expect to have lively discussions about emerging topics and challenges related to human aspects of software engineering. The industry participants will provide input on what topics are most important to focus on. We also expect that attendees will be able to identify potential collaborators for research projects and make connections to companies that they could otherwise not connect to. The collaborations will push the boundaries of empirical research and lead to high-profile publications. In addition, we expect to come up with guidelines and best practices for research in this domain, which can also be beneficial for industry.

Industrial impact. Industrial participants can benefit greatly from this seminar by learning and discussing existing state-of-the-art research as well as finding suitable potential collaborators.

Report Structure

The remainder of this report is structured as follows: First, we present the meetings schedule and short or extended abstracts of the invited talks of four researchers. Besides the individual 5-minute introductory presentations, these served as inspiration for the upcoming discussions during the Shonan meeting. Next, we provide summaries of the various discussions that we've had, either in break-out groups or the plenary. Finally, a list of participants as well as a summary, challenges and outcomes is provided.

Meeting Schedule

Check-in Day: March, 5, 2023 (Sun)

- Welcome Reception

Day 1: March, 6, 2023 (Mon)

- Opening
- Introduction
- Invited Talk #1: John Whittle
- Group Photo Shoot

Day 2: March, 7, 2023 (Tue)

- Invited Talk #2: Nicole Novielli
- Invited Talk #3: Andrew Bagel & Alexander Serebrenik
- Group sessions

Day 3: March, 8, 2023 (Wed)

- Group sessions
- Excursion and Main Banquet

Day 4: March, 9, 2023 (Thu)

- Group sessions
- Wrap up

Invited Talks

How can we equip software engineers to be proactive AI ethics proponents?

Jon Whittle, Data61

Software engineering has traditionally focused on how to build software reliably, affordably, safely and securely. This ignores a broader set of human values - such as inclusion, diversity, social responsibility, well-being and tradition etc. This talk argues that these broader human values should be embedded in software - since software fundamentally shapes society. The talk reported on six years of work in integrating human values in software engineering, and application of this work to responsible AI. The talk also reflected on the role of software engineering researchers in ensuring that AI systems are developed in a responsible way.

Towards Supporting Emotion Awareness of Software Developers

Nicole Novielli, University of Bari

Software development is an intellectual activity requiring creativity and problem-solving skills, which are known to be influenced by emotions. Developers experience a wide range of affective states during programming tasks, which may have an impact on their job performance and well-being. Early recognition of negative emotions, such as stress or frustration can enable just-in-time intervention for developers and team managers, in order to prevent burnout and undesired turnover. In this talk, I will present an overview of recent research findings of our empirical studies aimed at investigating the link between emotion and productivity, understanding the triggers for developers' emotions, and the strategies they implement to deal with negative ones and restore positive feelings. Furthermore, I will present the results of a field study involving software developers from five different companies investigating the feasibility of emotion recognition using a minimal set of non-invasive biometric sensors i.e. a wristband capturing the electrodermal activity and heart-related metrics. To conclude, I will be discussing open challenges and presenting early results of ongoing empirical studies.

Supporting Neurodiversity in Software Engineering

Andrew Begel, Carnegie Mellon University

My research aims to create the socio-technical infrastructure underpinning accessible technology and inclusive workplaces to provide opportunity, eliminate bias, and empower people with disabilities to fully engage and collaborate equitably with their non-disabled colleagues. My recent work aims to help neurodivergent individuals, who make up 15-20% of the world population with autism, ADHD, dyslexia, and others. We explored the challenges that neurodivergent developers and IT professionals face when using Microsoft's Azure

Portal and developed a set of neurodiversity-related user experience guidelines to help designers improve their software and make it easier to use by their neurodivergent customers. This case study illustrates the benefits and challenges of applying inclusive and universal design towards addressing the specific needs of neurodivergent users.

Gender and Age in Software Engineering

Alexander Serebrenik, Eindhoven University of Technology

In this talk I provide an overview of the results we have obtained when studying gender and age in software engineering.

While in our early work we have focused on activity of developers of different genders on such software engineering platforms as Stack Overflow and GitHub we have soon realised that focusing on individuals' behaviour is not enough and one has to consider teams, i.e., gender diversity within teams. A priori, diversity can have both benefits and drawbacks. Information-processing theory treats diversity as positive: bringing to the table a mixture of cultural/educational backgrounds, and access to different networks and broader information can enhance a team's creativity, adaptability, and problem solving skills; indeed diverse problem solving teams tend to outperform high-performing problem solving teams. This promise of better team performance is, however, threatened by communication challenges associated with diverse teams. Indeed according to the similarity-attraction theory of Williams and O'Reilly, people prefer working with others similar to them in terms of values, beliefs, and attitudes, as this facilitates communication between the team members. Similarly, social identity and social categorization theory of Tajfel postulates that people tend to categorize themselves into specific groups, and categorize others as outsiders; members of one's own group are then treated better than outsiders.

This is why in the following series of papers we have focused on communication in software development teams, and in particular, on patterns of suboptimal communication, so called "community smells" introduced by Tamburri et al. based on the management literature and observed in both open-source and closed-source software projects. We have established that community smells are related to code smells, i.e., suboptimal organisation of communication between developers corresponds to suboptimal organisation of the source code. For example, information overload represented by the black cloud co-occurs with lengthy methods and lone wolves, i.e., unsanctioned or defiant contributors who carry out their work irrespective or regardless of their peers, their decisions and communication, with ill-structured spaghetti code. Furthermore, in a follow up study we observed that gender diverse teams are less likely to develop black cloud since women are known to take mediating roles. Moreover, geographically dispersed team members are less likely to experience black cloud as well. A possible explanation is that managing people physically arranged in different parts of the world means using specific management tools and protocols for communication and collaboration, e.g., Trello and Jira which "nudge" the way of working towards a rather narrow, more disciplined approach reducing the noise associated with black cloud.

However, understanding the phenomenon is not enough - and this way we have joined several efforts aiming at increasing gender diversity in computing.

The first project consisted in translating and adapting US-based educational program “Beauty and Joy of Coding” to Dutch. We have opted for this educational program since it is known to be ‘programming-heavy’, on the one hand, and has been shown to attract non-traditional candidates such as women and non-CS majors to programming, on the other hand. At the moment the translation is being used and adjusted in one of the schools in the Netherlands. The second project consisted in organising women-centric hackathons in Brazil. Survey of the hackathon participants has shown they did not participate in such activities in the past due to not being confident enough in their technical skills or not being aware of the hackathons existence. The participants aimed at creating an interesting project during the hackathon but also had more long-term goals such as becoming part of the community or advancing their careers. It remains to be seen whether such hackathons really attract women to computing or whether their involvement remains limited to the hackathon itself. The two examples of trying to bring change in the gender composition of the computing world are of course only the first steps: it is imperative that we as researchers beyond understanding the experiences and needs of developers from minoritized group, to providing better support, either through better software engineering tools or through better software engineering processes.

Another diversity aspect we have been working is age. We have started by studying the public discourse surrounding age in software development. By analysing 24 articles published on blog platforms and news sites, we have observed that employability has emerged as the dominant theme. The employability strategies recommended by the articles included growing as a software engineer and changing the work environment: moving to a management role, and mastering modern technologies. The latter strategy is, however, controversial among developers on Hacker News: while some commenters indicate the importance of keeping up to date with technology development, others stress that learning new technology cannot counter the “cultural mismatch” but also that there “are skills past simply becoming proficient in new tools”. Furthermore, developers are being recommended to lower their expectation in terms of location, salary or job function aptly summarised by one of the articles as “you’re old, get over it”. The last group of strategies involved appearing young, which captures controversial strategies such as modifying one’s résumé to disguise age-related aspects as well as undergoing plastic surgery to look younger, an aspect that has recently been picked up by major US news outlets. This category further contains adopting patterns of youthful behaviour, including working overtime or during weekends, which are strategies known to conflict with other responsibilities such as family.

In the last part of the talk we have combined the two perspectives and discuss experiences of veteran women in software engineering as well as the “survival” strategies they have used. Having interviewed 14 study participants, we have identified the aforementioned strategies and experiences. Both among the strategies and among the experiences, in addition to the age-based experiences/strategies, gender-based experiences/strategies we have identified experiences/strategies related to age and/or gender. The latter category is telling: as it is often the case for individuals belonging to the intersection of diversity axes, they do not necessarily whether their experiences should be attributed to individual diversity axis or to the interplay of several such axes, i.e., veteran women are sometimes unsure whether the negative experiences were because of their gender or their age. There were not many Positive experiences related to age and gender, although Being a Role Model and More Opportunities Due to Gender and Age were found. One participant described how companies specifically looking to develop products aimed at her demographic led to opportunities: “A company approached me and said they were in the business, they wanted to make an app that would help predict who would have a stroke. . . They were like ‘our ideal candidate would be a Woman of Color [who has] also survived a stroke.’ ” Negative experiences were far more common, such as Seen as Non/less Technical , which has also been widely observed in the literature. We found that Gender Related Strategies contained the most strategies, with eight separate categories and 308 code segments. The categories were: Against Gender Bias Strategies, Career Related Strategies, Changing Work Environment, Changing Your Appearance, Communication Methods, Ignoring Situations, Traditionally Feminine, and Traditionally Masculine. Of these, Against Gender Bias Strategies was the largest category, with 70 code segments and eight subcodes, such as Backing Other Women Up.

Ultimately, the findings of our study have implications both for organisations employing or seeking to employ veteran women and for the veteran women themselves. Organisations should invest in creating a good working environment and a positive atmosphere, investing older developers of marginalized genders with sense of control of their work and their careers, supporting their promotion, assigning tasks and paying them on par with men. While these recommendations are true for any employer, they are even more pertinent for software engineering given the scarcity of older women and non-binary people in this industry. Moreover, specifically in case of software engineering the inclusion of developers who are more representative of the population ensures that the software can meet the needs of society as a whole. Developers themselves can move to a different work environment (e.g., by starting their company, moving to a different company, becoming consultant or manager) or try to change their work environment (e.g., by unionizing, standing up against gender bias or carving new opportunities for themselves). Changing appearance is one of the commonly mentioned but profoundly problematic strategies. We recognise that these recommendations are merely band aid solutions applied to systemic issues; however, we hope that in short-term they might help developers to survive in the industry.

And of course, veteran women are merely one of the intersections we might consider. Using survey data of U.S. STEM professionals (N = 25,324), Erin Cech examined whether white able-bodied heterosexual men (WAHM) are uniquely privileged in STEM. The results show that WAHM experience better treatment and rewards in STEM compared with members of all 31 other intersectional gender, race, sexual identity, and disability status categories. WAHM have been found to experience more social inclusion, higher professional respect, career opportunities, salaries and persistence intentions (compared to STEM professionals in other intersectional groups). This calls for additional studies taking the intersectional perspective and trying to understand experiences of software developers from additional intersectional groups.

Topic Summaries of Working Groups on Tuesday

After listening to the introductory presentations and invited talks, the group split into ad-hoc subgroups to start to define the scope of our ongoing discussions and collect initial ideas and challenges with respect to the Human Aspects of Software Engineering and the influence of AI on development processes and software engineers.

The group on **Developers' Interactions with AI** identified the following challenges and research opportunities:

- The general challenge is to identify appropriate and timely research questions, given that the technology is evolving so fast.
- Therefore, the group hypothesized that it's better to focus on long-lift problems such as impact, and less on the actual technology. They were also wondering about the balance of benefits and new issues that AI entails.
- In addition, they were wondering about how the workflow is impacted: are fewer people reviewing code? will the software development roles change (e.g. non-creative jobs might be reduced)? how will UX developers be impacted if interfaces are more and more chat-/voice-like? which new roles will emerge?

The next group discussed how the new future of work might impact **Collaboration and Well-Being**:

- One more general discussion was on improving our understanding of well-being by means of producing a more holistic list of factors that impact software engineer's well-being, creating definitions, and identifying ways to quantify well-being.
- There are factors that impact collaboration and well-being that cannot be influenced or directly acted upon, such as personality traits. These are harder to impact from the outside, whereas individuals can learn to take action on such things anytime.
- We were further wondering how much the individual's and company's culture influences collaboration and well-being. For example, we've learnt that in Japan people often take decisions in favor of their team, such as being able to get parental leave, but not taking them to keep supporting their team. Such cultural impacts need to be better understood and awareness increased, especially in multi-cultural teams. We were wondering if more explicitly describing a company's values or even describing them as rules (e.g. "you are explicitly encouraged to take your parental leave", "you may say no to meetings") could be helpful.
- When joining a team or company, there is often also a form of self-selection since most people prefer to work with other people who have a more similar way of working and thinking (e.g. a similar shared background), even though more diverse teams have been shown to perform better and have more diverse solutions.

- Teams and companies need to find a balance between bottom-up and top-down approaches to reach a certain degree of alignment of values (e.g. Schwartz model of basic human values).

- Finally, we've discussed opportunities of performing more cross-cultural studies to better support and foster the creation of diverse software development teams.

The third group started to discuss general themes and questions of when the topics of "AI & Software Engineering" and "Collaboration & Well-Being" are combined:

- Much of existing work has targeted individuals, so there is lots of room left to target teams, and better understand what a team really is and how it is defined.

- The team was further wondering how the huge changes that are currently happening in the space of AI and especially large language models (LLMs) impacts accountability (i.e., "who is responsible when the machine makes or decides things?") and our jobs (i.e., "how do these LLMs impact the required skill sets and how do we re-train people?")

- Concretely, for software developers, the team was wondering whether skills move to analyzing and quality reviewing code that was produced by the LLMs and no longer by human developers (i.e., editor vs creator)? The team agreed that these changes will definitely require more critical thinking of developers.

- Resulting questions for researchers are:
  - Which are the required skill sets of the software engineer of the near future?
  - Are we as researchers mindful of technological advances when designing studies?
  - What does software engineering mean in this new world of AI-generated software?
  - Could this reinforce the need for better program comprehension and visualization?
  - How is trust in LLMs generated when these models rely on AI-generated code, as opposed to e.g. Stack Overflow?

Topic Summaries of Working Groups on Wednesday and Thursday

Group 1: AI in SE

Members: Jon Whittle, Foutse Khomh, Bin Lin, Christoph Treude, Yasutaka Kamei, Masanari Kondo, Daniel German, Michele Lanza

The discussion notes and tables were inserted to ChatGPT to summarize, after which the output was cross-checked:

Generative AI, such as large language models (LLMs), has the potential to impact various phases of the software development lifecycle, including requirements, design, implementation, testing/validation, maintenance, and procurement. In the requirements phase, generative AI can automate several tasks, such as summarizing requirements elicitation documents, rapid prototyping, and ensuring consistency in requirements documents. However, eliciting requirements is fundamentally a human activity that may not be easy to automate.

Similarly, in the design phase, generative AI can automate some tasks, such as good for simple designs, including design patterns, and visualizing designs. However, the abstraction, which is fundamental for design, may be hard to automate, requiring human intervention.

In the implementation phase, generative AI can automate code review and rapid prototyping, making human developers obsolete. However, generated code can be hard to understand because of inadequate naming conventions, which presents a problem that requires human intervention.

In the testing/validation phase, generative AI can automate test-driven development, improving the efficiency and effectiveness of software development.

In the maintenance phase, generative AI can automate many tasks, such as auto-documentation, human-readable explanations, and reverse engineering, but some maintenance tasks may still require human intervention.

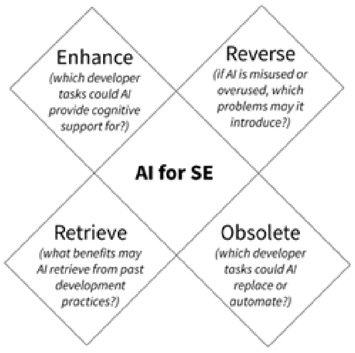
While generative AI can improve the efficiency and quality of software development, it also raises concerns around ethics, bias, privacy, and security. The use of generative AI in software engineering requires careful consideration of these ethical implications to ensure that the technology is used in a responsible and ethical manner.

Moreover, the economic implications of implementing and maintaining generative AI need to be carefully evaluated, as there may be costs associated with implementing and maintaining the technology. The successful integration of generative AI into software engineering will depend on a careful balance between the benefits of the technology and the potential risks and costs. Therefore, the optimal approach is likely to be a combination of generative AI and human developers, leveraging the benefits of generative AI while mitigating the potential risks and costs. Ultimately, the responsible and ethical use of generative AI in software engineering can lead to more efficient and effective software development while minimizing the potential risks and costs associated with the technology.

Group 2: AI in SE

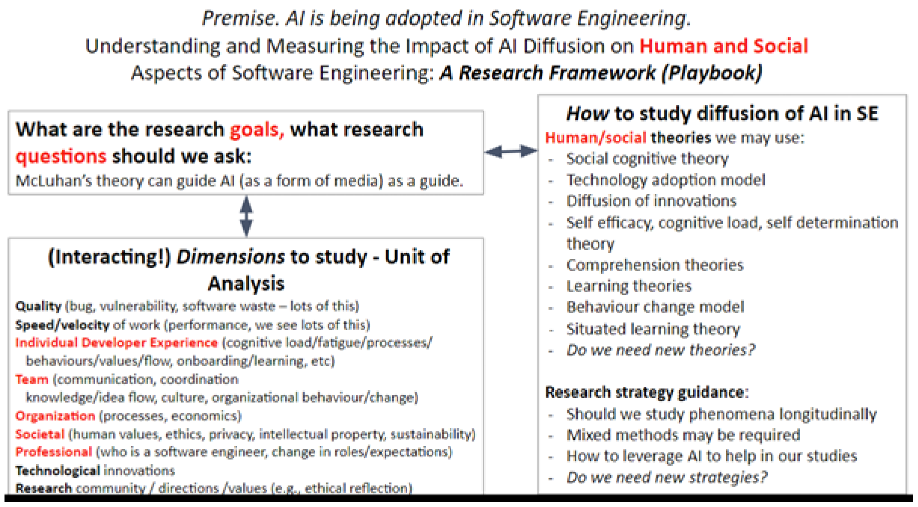
Members: Dong Wang, Daniel Russo, Nicole Novielli, Takashi Kobayashi, Margaret-Anne Storey

This break-out group started developing a framework for understanding and measuring the impact of AI diffusion on Human and Social Aspects of Software Engineering, based on the perspectives of McLuhan's Tetrad (see Figure 1). The outcomes, including examples of how to apply the framework, are summarized in Figure 2 - 7.



Enhance
(which developer tasks could AI provide cognitive support for?)
Reverse
(if AI is misused or overused, which problems may it introduce?)
AI for SE
Retrieve
(what benefits may AI retrieve from past development practices?)
Obsolete
(which developer tasks could AI replace or automate?)

Figure 1: McLuhan's Tetrad.



Premise. AI is being adopted in Software Engineering.
Understanding and Measuring the Impact of AI Diffusion on Human and Social Aspects of Software Engineering: ***A Research Framework (Playbook)***

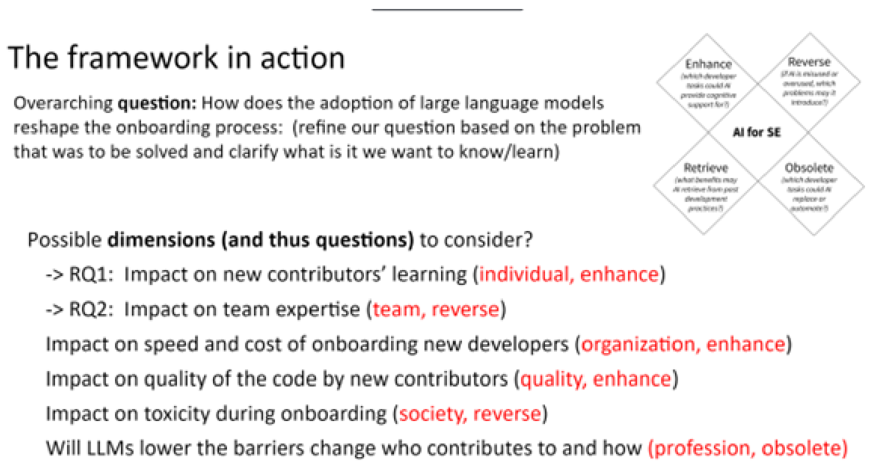
What are the research goals, what research questions should we ask:
McLuhan's theory can guide AI (as a form of media) as a guide.

(Interacting!) *Dimensions* to study - Unit of Analysis
Quality (bug, vulnerability, software waste – lots of this)
Speed/velocity of work (performance, we see lots of this)
Individual Developer Experience (cognitive load/fatigue/processes/ behaviours/values/flow, onboarding/learning, etc)
Team (communication, coordination knowledge/idea flow, culture, organizational behaviour/change)
Organization (processes, economics)
Societal (human values, ethics, privacy, intellectual property, sustainability)
Professional (who is a software engineer, change in roles/expectations)
Technological innovations
Research community / directions /values (e.g., ethical reflection)

***How* to study diffusion of AI in SE**
Human/social theories we may use:
- Social cognitive theory
- Technology adoption model
- Diffusion of innovations
- Self efficacy, cognitive load, self determination theory
- Comprehension theories
- Learning theories
- Behaviour change model
- Situated learning theory
- *Do we need new theories?*

Research strategy guidance:
- Should we study phenomena longitudinally
- Mixed methods may be required
- How to leverage AI to help in our studies
- *Do we need new strategies?*

Figure 2: Research Framework for Understanding and Measuring the Impact of AI Diffusion on Human and Social Aspects of SE.


Enhance
(which developer tasks could AI provide cognitive support for?)
Reverse
(if AI is misused or overused, which problems may it introduce?)
AI for SE
Retrieve
(what benefits may AI retrieve from past development practices?)
Obsolete
(which developer tasks could AI replace or automate?)

The framework in action

Overarching **question:** How does the adoption of large language models reshape the onboarding process: (refine our question based on the problem that was to be solved and clarify what is it we want to know/learn)

Possible **dimensions (and thus questions)** to consider?

-> RQ1: Impact on new contributors' learning (individual, enhance)

-> RQ2: Impact on team expertise (team, reverse)

Impact on speed and cost of onboarding new developers (organization, enhance)

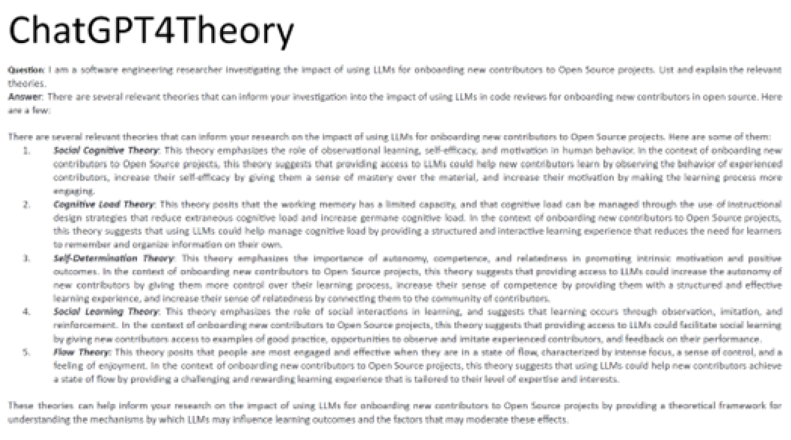
Impact on quality of the code by new contributors (quality, enhance)

Impact on toxicity during onboarding (society, reverse)

Will LLMs lower the barriers change who contributes to and how (profession, obsolete)

Figure 3: The Framework in Action.

]



ChatGPT4Theory

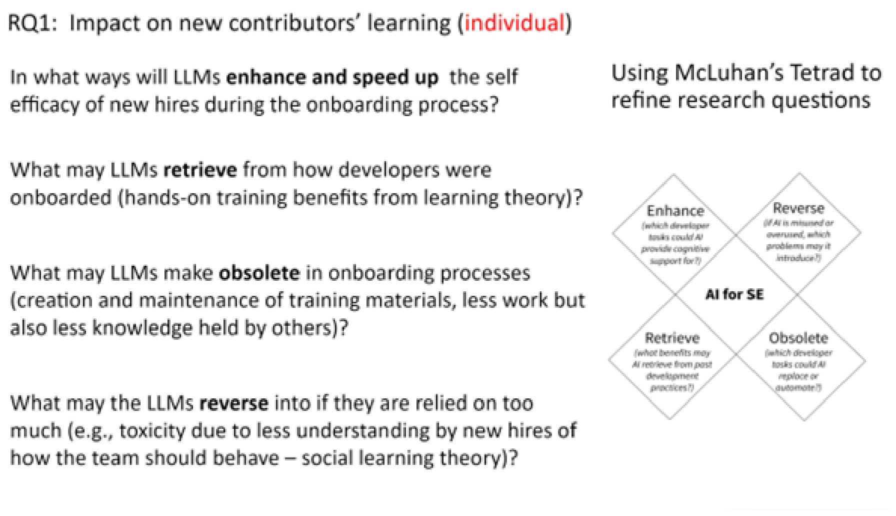
Question: I am a software engineering researcher investigating the impact of using LLMs for onboarding new contributors to Open Source projects. List and explain the relevant theories.
Answer: There are several relevant theories that can inform your investigation into the impact of using LLMs in code reviews for onboarding new contributors in open source. Here are a few:

There are several relevant theories that can inform your research on the impact of using LLMs for onboarding new contributors to Open Source projects. Here are some of them:

1. ***Social Cognitive Theory***: This theory emphasizes the role of observational learning, self-efficacy, and motivation in human behavior. In the context of onboarding new contributors to Open Source projects, this theory suggests that providing access to LLMs could help new contributors learn by observing the behavior of experienced contributors, increase their self-efficacy by giving them a sense of mastery over the material, and increase their motivation by making the learning process more engaging.
2. ***Cognitive Load Theory***: This theory posits that the working memory has a limited capacity, and that cognitive load can be managed through the use of instructional design strategies that reduce extraneous cognitive load and increase germane cognitive load. In the context of onboarding new contributors to Open Source projects, this theory suggests that using LLMs could help manage cognitive load by providing a structured and interactive learning experience that reduces the need for learners to remember and organize information on their own.
3. ***Self-Determination Theory***: This theory emphasizes the importance of autonomy, competence, and relatedness in promoting intrinsic motivation and positive outcomes. In the context of onboarding new contributors to Open Source projects, this theory suggests that providing access to LLMs could increase the autonomy of new contributors by giving them more control over their learning process, increase their sense of competence by providing them with a structured and effective learning experience, and increase their sense of relatedness by connecting them to the community of contributors.
4. ***Social Learning Theory***: This theory emphasizes the role of social interactions in learning, and suggests that learning occurs through observation, imitation, and reinforcement. In the context of onboarding new contributors to Open Source projects, this theory suggests that providing access to LLMs could facilitate social learning by giving new contributors access to examples of good practice, opportunities to observe and imitate experienced contributors, and feedback on their performance.
5. ***Flow Theory***: This theory posits that people are most engaged and effective when they are in a state of flow, characterized by intense focus, a sense of control, and a feeling of enjoyment. In the context of onboarding new contributors to Open Source projects, this theory suggests that using LLMs could help new contributors achieve a state of flow by providing a challenging and rewarding learning experience that is tailored to their level of expertise and interests.

These theories can help inform your research on the impact of using LLMs for onboarding new contributors to Open Source projects by providing a theoretical framework for understanding the mechanisms by which LLMs may influence learning outcomes and the factors that may moderate these effects.

Figure 4: Asking ChatGPT to list and explain relevant Theories.


RQ1: Impact on new contributors' learning (individual)

In what ways will LLMs **enhance and speed up** the self efficacy of new hires during the onboarding process?

What may LLMs **retrieve** from how developers were onboarded (hands-on training benefits from learning theory)?

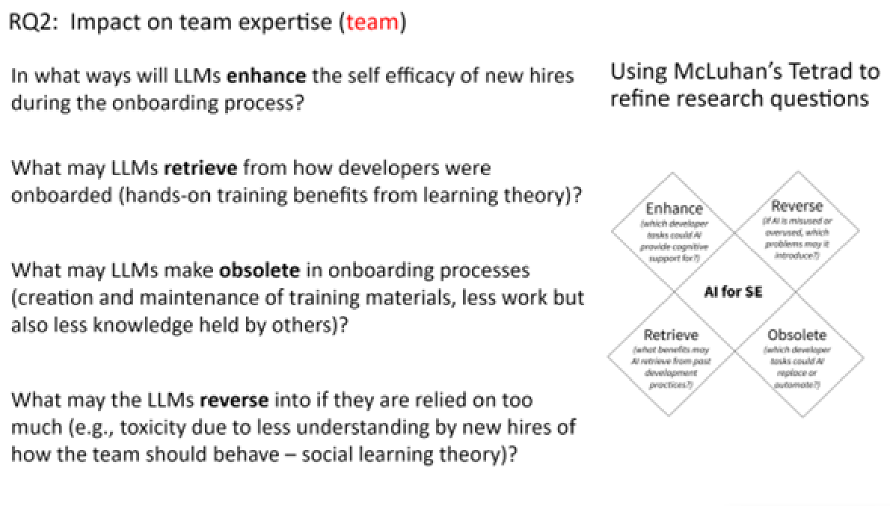
What may LLMs make **obsolete** in onboarding processes (creation and maintenance of training materials, less work but also less knowledge held by others)?

What may the LLMs **reverse** into if they are relied on too much (e.g., toxicity due to less understanding by new hires of how the team should behave – social learning theory)?

Using McLuhan's Tetrad to refine research questions

Enhance
(which developer tasks could AI provide cognitive support for?)
Reverse
(if AI is misused or overused, which problems may it introduce?)
AI for SE
Retrieve
(what benefits may AI retrieve from past development practices?)
Obsolete
(which developer tasks could AI replace or automate?)

Figure 5: Impact on new contributors' learning (individual perspective).


RQ2: Impact on team expertise (team)

In what ways will LLMs **enhance** the self efficacy of new hires during the onboarding process?

What may LLMs **retrieve** from how developers were onboarded (hands-on training benefits from learning theory)?

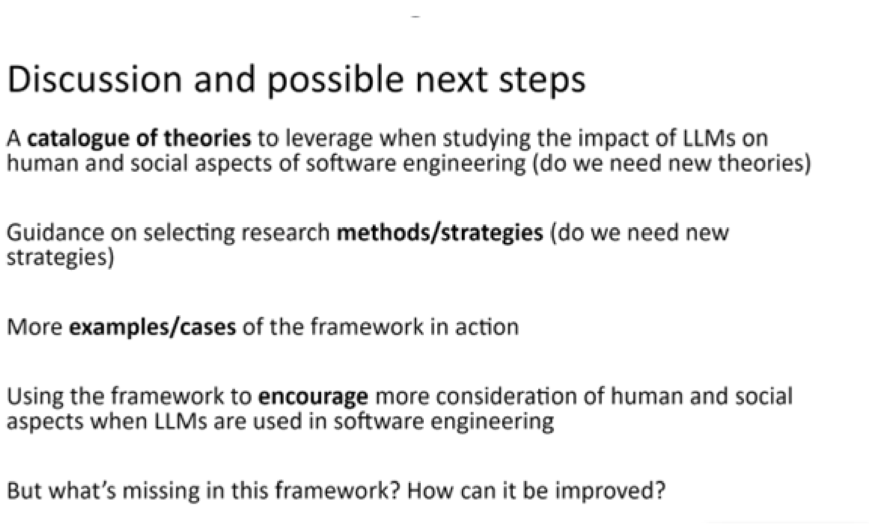
What may LLMs make **obsolete** in onboarding processes (creation and maintenance of training materials, less work but also less knowledge held by others)?

What may the LLMs **reverse** into if they are relied on too much (e.g., toxicity due to less understanding by new hires of how the team should behave – social learning theory)?

Using McLuhan's Tetrad to refine research questions

Enhance
(which developer tasks could AI provide cognitive support for?)
Reverse
(if AI is misused or overused, which problems may it introduce?)
AI for SE
Retrieve
(what benefits may AI retrieve from past development practices?)
Obsolete
(which developer tasks could AI replace or automate?)

Figure 6: Impact on team expertise (team perspective).



Discussion and possible next steps

A **catalogue of theories** to leverage when studying the impact of LLMs on human and social aspects of software engineering (do we need new theories)

Guidance on selecting research **methods/strategies** (do we need new strategies)

More **examples/cases** of the framework in action

Using the framework to **encourage** more consideration of human and social aspects when LLMs are used in software engineering

But what's missing in this framework? How can it be improved?

Figure 7: Main points of discussion and possible next steps.

Group 3: Future of Work in SE

Members: Jin Guo, Hideaki Hata, Alexander Serebrenik, Thomas Zimmermann, Gail C. Murphy, Filippo Lanubile

Discussion Scope

- Well-being, inclusive, Organizational or Software developers,
- The concept of software development
- Skills for software developers for future – Is everyone a developer?
- Data scientists and other roles are part of the team producing large software systems.
- Considering systems being built, and the requirement of qualities for the people who are building it.
- Skill shortage, is it still the case in the future.
- Professional software engineers will work differently in the future – program comprehensive, debugging.
- In the future, are we moving away from code? Model-driven software development is the same idea but never works. We cannot parameterize enough to get the quality we want. We still have a long way to go to dedicate design to LLM.
- Open source and sponsorship. Is this model the future? If LLM starts charging big fees and causes barriers for the open source community. Business opportunities for supporting those communities?
- Responsibility, the people right at the front line have the largest responsibility. Remote work also puts more emphasis on responsibility. Trust.
- Machines write code, humans do quality assurance. How to support those paradigm changes?
- What does community look like? What should they share? Responsibility? When to take it and when to push away?

Challenges

Categorized with Labels: [IMP] Impact of LMMs on source code; [PROC] Development process; [QA] QA; [SKI] Role/Skills; [COM] Community; [ECO] Economic; [LEG] Legal

- Who is responsible? For what? [SKI]
- Finding tradeoffs between people and service expenses? (cost-sharing)
- What is good code? For input to models and also output to people.
- How can we ensure inclusive code?

- Provenance: Where are things coming from? [SKI]
- What new responsibilities must a developer assume? [SKI]
- How does more responsibility affect well-being? What training do we need? [SKI]
- What are acceptable users for LLM? How to choose an LLM?
- How to balance the responsibility between individuals and community?
- How to deal with code clones generated by LLM? (Vulnerability tracking, Clone Management)
- Disruption of abstraction
- AI tends to exuberate existing problems (e.g. keep things less inclusive)?
- Impact of LLMs on technical debt (e.g., many clones, poor architecture)?
- What is the process of maintenance and evolution? (LLM evolve, LLM remains the same, but you want to change the code, additional curated code)
- How do we maintain the ecosystem? What does the ecosystem become? [COM]
- What to find which part has to be changed when the context of the system changes?
- What is a bug? How do we report bugs?
- What are the blind spots for testing?
- What tools do we need to deal with the uncertainty of LLM generated code (e.g., interrogation)? [SKI]
- How do we program in dialog (and maintain state)? [PROC]
- Will legacy be no longer a problem, and become easy to innovate? Or will we create more legacy? [COM] [PROC] [ECO]
- How will LLM change skill development? [SKI]
- How do we prevent adversarial attacks on the LLM? [QA]
- Will the developers be limited in career choices? [SKI]

Group 4: Future of Work in SE

Members: Andrew Begel, Kelly Blincoe, Thomas Fritz, Reid Holmes, Andre N. Meyer, Raula G. Kula

Two Visions for the Future of Work in SE

One vision focuses on the future of **effective teams**, which is to...

- ... ensure effective teaming in the future, we will need to understand the effects of team diversity (e.g., gender, culture, ethnicity, disability, values, etc) on communication, collaboration, coordination, productivity, motivation, happiness at the individual, team, and organizational levels.
- ... educate the next, more democratized, generation of software engineers with new tools, new processes, different skills, and values necessary to build software for the ever-changing needs of society.
- ... foster effective, sustainable, inclusive, and democratized teamwork we need to better understand how to evolve and support the social skills that software development teams need to thrive in the new future of work.

The other vision focuses on increasing the focus on essential complexity, by **raising the abstraction levels in software developments and its Team members**:

- To effectively reason and understand the higher levels of generated code, so that teams can easily express their needs without losing trust of the generated code.
- Incorporating the non-deterministic nature of generated code.
- Training specialized members to piece and glue code together, connect the dots, scale software, and optimize software and personalize it.

With the advent of large language models like ChatGPT and Copilot, the problem that we are being faced with is the need for a better understanding of how people are involved in software development.

Vision 1: Effective interaction of development teams in hybrid work requires evolving existing social skills

We need different skills, different tools, different processes, better educational experiences for hybrid work. We define the following research questions to help tackle the vision:

Individual Work

- What could be a good work schedule of a software engineer in the future? (personal preferences, chronotypes, family situation, etc.)
- Why might we not work less even though we could save lots of time with AI in the future? (economic, social pressure, value driven)
- We had this shift from remote to hybrid work; what could other similar shifts be in the near future? (e.g. less work hours and work routines)

Team

- What are social skills required to be good at hybrid work, and how do they differ from in-person work?
- In which ways is hybrid work making teamwork more difficult?
- What is the impact of hybrid or remote-only work on team cohesion and loyalty towards the team and company?
- How should we operationalize metrics for team communication effectiveness in order to measure them? (based on SPACE framework)

Organizational Issues

- How can we train teaming skills? Which ones are teachable and which ones are more innate?
- How do you organize your teams at an organizational level to be effective when working hybrid?
- Are values determined bottom up or are they determined from the top?
- How does the governance structure employed by the team affect the adoption of values?
- Is it possible to change toxic cultures from below or are individual workers powerless?

Diversity, Equity, Inclusion, Belonging

- How do we recruit, retain and advance more women into SE?
- Does the introduction of generative AI for code creation. support or reduce inclusion, diversity, biases?
- How can we increase the diversity and inclusion of marginalized groups (additional cultures, ethnicities) into software and what is their effect on the team and product? How do we make those values more explicit and deliberately chosen?

Longitudinal

- How has teaming changed over the past 60 years? How do teams break down differently now than before?
- What are the well-being factors of an individual, team and organization? What model describes these factors?
- How do you define a team culture at the beginning when no one really knows one another, knows one another's values, skills, or contributions?
- Does the communication culture and needs stay the same over time?
- How do you evolve communication processes and practices and culture over time?

We expect the following **outcomes** from tackling the aforementioned research questions:

- Create a team **norming framework** for negotiating, evaluating, evolving teamwork, values, cultures as the team's needs change.
- Building on the SPACE framework, collect a **set of operationalized metrics** to create a holistic understanding of teamwork effectiveness and efficiency and show how this has changed over time.
- Create a **longitudinal chart** of how effective teamwork skills have evolved over the long history of software development.
- Create a set of **pedagogical modules** to teach teamwork for the next generation of developers and the way that they prefer to work.
- **Assessment of existing pedagogy** for software teamwork to identify they need to change to support the needs of the future of work.

Vision 2: Increased Focus on Essential Complexity in Software Engineering

Changes in Abstraction Levels. We have seen changes in abstraction levels throughout the history of software engineering. Assembly languages abstracted details of the hardware systems. High level programming languages and compilers allowed us to describe/program systems closer to the mental model humans have and abstract away from the details of the processing unit of the computer. What we are seeing now is another change in the abstraction level. Large language models are now allowing us to express our needs in natural language and provide boilerplate software code for common programming tasks. This reduces the need to fill in tedious syntax, but the most challenging parts of software engineering still remain and are even more emphasized.

Increased Non-Determinism and Complexity. With the increase in the complexity of systems, their high interdependence with other modules, and especially the use and integration of ML models in the systems and their development, we now shifted to an era in which systems are more non-deterministic than ever. This excels the need for a better reasoning about the systems we built, explainability and trust of the system, and ensuring certain "values" and properties in the system, such as its fairness. At the same time the increasing complexity of the systems also requires a stronger focus on properly capturing what the user wants in the first place and ensuring the right user experience, as well as correctly designing and integrating the systems.

Specification and User Experience (User - Developer Interaction). Snippets of code do not make usable software systems. Software systems are complex and their usefulness does not come down to only the quality of the written source code. Software engineers must understand if the system being built is the right one. Will the system solve a problem for the people who are using it? Will the people who use the system feel good when using the system? Will they want to keep using it? There are essential elements of the human experience that require human understanding to ensure high quality software.

Design. Large language models cannot generate entire software systems. Throughout the history of software engineering, integration of individual pieces

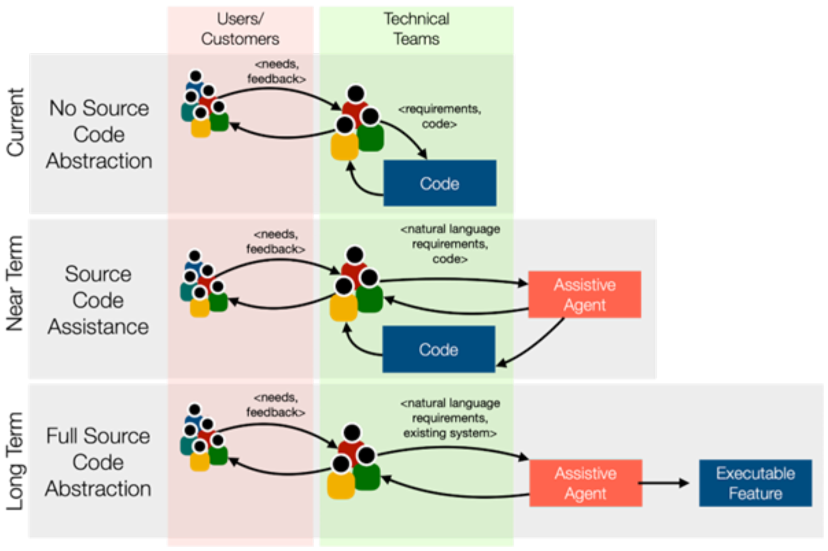


Figure 8: Visualization of current, near term and long term integration of Generative AI models and their impact on Source Code Abstraction.

of software into full software systems has been a challenge. Integration is extremely precise, tying the pieces together, multiple developers together specifying "values"

Reasoning about generated code. Another element that still requires human understanding is the reasoning around the code that is being produced by these large language models. The prior changes in abstraction levels in software engineering have been deterministic ones. If you give the same software code to the same compiler, it will always produce the same output. This is not the case with large language models. The models are trained from millions of examples of prior code and each time you make a request, new and different code will emerge.

List of Participants

- Thomas Fritz, University of Zurich (Organizer)
- Yasutaka Kamei, Kyushu University (Organizer)
- Thomas Zimmermann, Microsoft (Organizer)
- André N. Meyer, University of Zurich (Supporting organizers)
- Raula Kula, Nara Institute of Science and Technology
- Jin Guo, McGill University
- Hideaki Hata, Shinshu University
- Alexander Serebrenik, Eindhoven University of Technology
- Kelly Blincoe, University of Auckland
- Daniel German, University of Victoria
- Reid Holmes, University of British Columbia
- Michele Lanza, Software Institute - USI, Lugano, Universita della Svizzera italiana
- Gail C. Murphy, University of British Columbia
- Bin Lin, Radboud University
- Margaret Storey, University of Victoria
- Dong Wang, Kyushu University
- Masanari Kondo, Kyushu University
- Andrew Begel, Carnegie Mellon University
- Jon Whittle, CSIRO's Data61
- Christoph Treude, The University of Melbourne
- Foutse Khomh, Polytechnique Montréal
- Daniel Russo, Aalborg University
- Takashi Kobayashi, Tokyo Institute of Technology
- Nicole Novielli, University of Bari
- Filippo Lanubile, University of Bari

Executive Summary

The Shonan meeting resulted in several very fruitful and lively discussions, with a focus on creating an initial list of challenges and opportunities for research in the area of and intersection of "Artificial Intelligence" (especially AI-generated software), the "New Future of Work" (especially teamwork & collaboration) and "Human Aspects of Software Engineering" (especially inclusivity and ethical values).

Overall, the discussions highlighted the need for ongoing research and understanding of the complex interactions between AI, software engineering, and human aspects, including collaboration, well-being, cultural influences, and the evolving roles of software developers in the context of technological advancements. It also emphasized the importance of considering long-term impacts and ethical considerations in this rapidly evolving field.

In what follows, we first summarize **key challenges and opportunities for research**:

- **Impact of AI on workflow and job roles**: The group discussed how AI may affect the roles and responsibilities of software developers, such as potentially reducing the need to write large chunks of code, but highly increasing the value of code reviews. One group suggests to increasing focus on essential complexity in software engineering by considering changes in the **abstraction levels of software development**, e.g., by enabling software engineers to express their needs in natural language and generate code for common programming tasks. Nonetheless, software engineering still requires reasoning and understanding the higher levels of generated code, incorporating non-deterministic nature of generated code, training specialized members, scaling software, optimizing software, and personalizing it.

- The group discussed the need to understand the **impact of AI and large language models (LLMs) on teams, accountability, and job roles**. They raised questions about the skills required for software engineers in the future, the implications for research design, the meaning and future of software engineering in the context of AI-generated software, and the increasing importance of program comprehension and visualization to conduct thorough code reviews that reduce risks. The group also touched upon the issue of generating trust and responsibility when relying on AI-generated code.

- The group further discussed the potential **impact of generative AI (esp. LLMs) on various phases of the software development lifecycle**, including requirements, design, implementation, testing/validation, maintenance, and procurement. They highlight that while generative AI can automate several tasks in these phases, such as summarizing requirements, automating code reviews, and auto-documentation, there are limitations to fully automating human activities such as eliciting requirements and abstraction of design. They also note that generated code may be hard to understand due to inadequate naming conventions, requiring human intervention and code reviews. Additionally, the ethical implications

of using generative AI in software engineering, such as bias, privacy, and security, need to be carefully considered and addressed.

- Relatedly, one group identified more specific and additional **challenges that research and industry need to tackle in the context of LLM-generated code**, such as responsibility, criteria for good code, ensuring inclusivity and reducing biases in code, understanding provenance, developer and user training, as well as managing code quality, security and legacy.

- One breakout group initiated the development of a **framework for understanding and measuring the impact of AI diffusion on human and social aspects of software engineering**, by considering the diffusion based on the perspectives of McLuhan's Tetrad. They also provide examples of how to apply the framework.

- Researchers' and practitioners' challenges in keeping up with the **rapid evolution of AI-based technology**: The group highlighted the difficulty in keeping up with the rapid pace of technological advancements in AI and software engineering, and emphasized the need to focus on long-term problems and impacts rather than just the technology itself.

- The researchers agreed that **successful software development teams need to consider core human values**, including inclusion, diversity, social responsibility, emotions and well-being. These teams will, in most cases, yield better and more creative solutions that are required to keep up with today's requirements and pace. Therefore, training and awareness increase is required to foster effective communication and collaboration within these diverse teams.

- The researchers explored **factors that impact collaboration and well-being among software engineers** in the future of work, such as personality traits, company culture, and team dynamics. They discussed the need for a holistic understanding of these factors, the influence of culture on decision-making, and the balance between bottom-up and top-down approaches in aligning team values.

Finally, the working group summaries contain lists of very specific **research questions** that the community aims to tackle in the short as well as longer term. In addition, concrete next steps include the creation of a **vision paper** to discuss the ideas, challenges and opportunities in a short paper. In addition, one group is considering writing a **paper on their framework** for understanding and measuring the impact of AI diffusion on human and social aspects of SE. Finally, Daniel Russo is currently in the planning phase of organizing a **follow-up workshop in Denmark**, that will also be held in the Dagstuhl/Shonan-format.
The organizers also wanted to thank all attendees for their participation, as well as the National Institute of Informatics for supporting this wonderful Shonan meeting.