ISSN 2186-7437

**NII Shonan Meeting Report**

No. 176

# Foundation Models and Software Engineering: Challenges and Opportunities

Zhen Ming (Jack) Jiang
Ahmed E. Hassan
Yasutaka Kamei

March 25–28, 2024


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

# Foundation Models and Software Engineering: Challenges and Opportunities

Organizers:
Zhen Ming (Jack) Jiang (York University, Canada)
Ahmed E. Hassan (Queen's University, Canada)
Yasutaka Kamei (Kyushu University, Japan)

March 25–28, 2024

## 1 Introduction and Background

Generative AI is an AI technology which is capable of generating text, images, or other media, using generative models. According to Gartner's Hypercycle, the generative AI is currently on the peak of the inflated expectations on emerging technologies [11]. Its market is expected to have an explosive growth in the next ten years a compound annual growth rate (CAGR) of 40% from $40 billion in 2022 to $1.3 trillion in 2033 [9]. This is mainly driven by the increasing number of adoptions for Foundation Models (FM) and their associated use cases (e.g., OpenAI's ChatGPT and GitHub's Copilot). FMs are trained on a huge set of data and can be adapted (e.g., fine-tuned) to a wide range of downstream tasks [13]. Currently the most popular type of FM is Large Language Models (LLMs), which is trained on large corpus of unlabelled text data. Although these LLMs can perform various tasks from language translation to code generation, there are a few serious drawbacks which prevent FMs to be used alone as a general purpose framework in a much wider usage context. Instead of simply waiting for FMs to resolve these issues and become general purpose, current industry and academic researchers have come up with various innovative engineering solutions/frameworks to mitigate these issues. Below we describe a few such examples:

- **Grounding**: Hallucinations refer to the problem of FMs sometimes generating texts that is incorrect or purely fictional [18]. To mitigate this issue, it is recommended to incorporate FMs with some domain specific knowledge databases, such as a vector database or a knowledge graph, so that the answers can be semantically searched within these right context.
- **External Tool Uses**: benchmarking studies have shown that FMs are not good at logical reasons tasks such as mathematical calculations [6] and logical inference [12]. On the contrary, traditional software applications are coded based on rules and excels in reasoning. Hence, various solutions have enabled LLMs to invoke external tools [20] or third-party plugins [3].
- **Prompt Engineering**: Interacting LLMs are quite different from traditional ML models or classic software applications. Users write natural

language like instructions, called prompts, to the LLMs. However, LLMs may output incorrect or suboptimal results due to misunderstanding of the intensions behind these prompts. Hence, a new discipline, called prompt engineering [5], which focuses on developing tools and techniques to optimize the use of prompts for LLMs to accomplish a variety of tasks.

In addition to the aforementioned engineering techniques, various engineering frameworks (e.g., Langchain [18], HuggingGPT [21], and AutoGPT [2]) have been developed to facilitate engineers to better FM-powered applications. These applications interact with one or multiple FMs and interacts with third party tools/frameworks. Hence, they are more capable of completing more complex tasks like price matching and enterprise search.

Researchers and practitioners believe that the above synergy between the software engineering (SE) and FM is just the beginning and needs to continue throughout this new era of generative AI. This is mainly due to the following three reasons: (1) FMs are black-boxes which require researchers and practitioners to explore and experiment in various ways to uncover their emergent behaviors and drawbacks. On one hand, newly discovered emergent capabilities (e.g., chain-of-thought reasoning and instruction following [23]) are reported for FM models. On the other hand, through experimentation and trail-and-error, new risks and drawbacks associated will gradually be revealed and reported. (2) New types of FM models (e.g., multimodality FM models [10] and world models [8]) or domain specific FM models (e.g., FMs in Finance [24] and IT operations [15]), which equipped with new or enhanced capabilities, are being proposed at a much faster pace with new capabilities. (3) As these FM-powered software applications are slowly moving from research labs into production, in addition to technology novelty, additional concerns (e.g., legal and trustworthy concerns, costs and efficiency, etc.) need to be properly evaluated and addressed before the formal product launch. Failure to address these concerns will result in profit loss or even blockage of product sales. This meeting brought together leading researchers to discuss current and future trends and challenges related to FMs in and for SE. It marks the first Shonan seminar focused on the topic of FM and SE, which has the potential to transform the field of SE. In the remain part of this report, we will use FMs and LLMs interchangably.

## 2 Meeting Overview

Shonan Seminar #176 focuses on the challenges and opportunities of Foundation Models and Software Engineering. It took place between March 25 and March 28, 2024. The whole seminar is discussion-oriented, which consists of one keynote and three break out discussion sessions. Dr. Ahmed Hassan opened the seminar with his talk regarding "AIware in the Foundation Models Era". Then professor Lingming Zhang from the University of Illinois Urbana-Champaign in USA delivered his keynote, titled "Software Quality Assurance in the Era of Large Language Models". The participants spent the remaining morning of the first day briefly introduced themselves and their research interests. The remaining seminar is broken down into three break out sessions, which consists of a total of nine different discussion topics. Before each break out session, everyone meets up and decides on two to four topics to discuss during each session based on the relevance, importance, and interests among the participants. Each decided topic was assigned with one discussion lead and one scriber. After each break out session, the discussion lead presented a summary of the discussions for each of the topic. Below are the list of topics which were discussed:

- During the first discussion session, the following two topics were discussed: *Code Generation* and *Productivity*.
- During the second discussion session, the following three topics were discussed: *Data Quality*, *FM-powered agent-oriented software engineering*, and *FM-powered software testing*.
- Dring the third discussion session, the following three topics were discussed: *Process*, *Education*, and *Synergizing classical SE analysis and FM*.

## AIware in the Foundation Models Era

Ahmed E. Hassan, Queen's University

"Software for all and by all" is the future of humanity. AIware, i.e., AI-powered software, has the potential to democratize software creation. The definition of software along with many Software Engineering (SE) aspects, processes, tools, platforms, and techniques will need to be either reimagined, reformulated or redesigned, enabling individuals of all backgrounds to participate in its creation with higher reliability and quality. Over the past decade, software has evolved from human-driven Codeware to the first generation of AIware, known as Neuralware, developed by AI experts. Foundation Models (FMs, including Large Language Models or LLMs), ushered in software's next generation, Promptware, led by domain and prompt experts. However, this Promptware merely scratches the surface of software's future. We are already witnessing the emergence of the next generation of software, Agentware, in which humans and intelligent agents jointly lead the creation of software. With the advent of brain-like World Models and brain-computer interfaces, we anticipate the arrival of Mindware, representing the 5th generation of software. Agentware and Mindware promise greater autonomy and widespread accessibility, with non-expert individuals, known as Software Makers, offering oversight to autonomous agents.

In this talk, Dr. Hassan explained why the SE community will need to develop fundamentally new approaches and evolve existing ones, so they are suitable for a world in which software creation is within the reach of Software Makers of all levels of SE expertise, as opposed to solely expert developers. We must recognize a shift in where expertise lies in software creation and start making the needed changes in the type of research that is being conducted, the ways that SE is being taught, and the support that is offered to software makers. Additionally, Dr. Hassan also discussed several initiatives aimed at building a collaborative community to work collectively toward this vision. He reported progress on efforts such as the FM+SE Vision 2030 event [4] and the First AIWare Conference [1].

## Software Quality Assurance in the Era of Large Language Models

Lingming Zhang, University of Illinois Urbana-Champaign, USA

Abstract: In recent years, Large Language Models (LLMs), such as GPT-4 and Claude-3, have shown impressive performance in various downstream applications, including software engineering. In this talk, I will discuss the potential impact of modern LLMs on the important problem of software quality assurance, along with our recent research findings. I will first talk about the new opportunities and possibilies LLMs can offer for better quality assurance of real-world software systems. Next, I will talk about the new quality assurance challenges or issues raised by code LLMs themselves and deep learning in general, including strategies for mitigation. Lastly, I will also briefly discuss our recent experiences in building fully open-source code LLMs (such as StarCoder2 and Magicoder) for supporting better software quality assurance in the LLM era.

# 3 List of Participants

- Alexandar Serebrenik, Eindhoven University of Technology, Netherlands
- Gail Murphy, University of British Columbia, Canada
- Andreas Zeller, CISPA Helmholtz Center for Information Security, Germany
- Cor-Paul Bezemer, University of Alberta, Canada
- Masanari Kondo, Kyushu University, Japan
- Bowen Xu, North Carolina State University, USA
- Qinghua Lu, CSIRO, Australia
- Ipek Ozkaya, Carnegie Mellon University, USA
- Hironori Washizaki, Waseda University, USA
- Foutse Khome, Polytechnique Montreal, Canada
- Lionel Briand, University of Ottawa, Canada
- Satish Chandra, Google, USA
- Romain Robbes, University of Bordeaux, France
- Maliheh Izadi, Delft University of Technology, Netherlands
- Denys Poshyvanyk, William and Mary, USA
- Ying Zou, Queen's University, Canada
- David Lo, Singapore Management University, Singapore
- Mei Nagappan, University of Waterloo, Canada
- Shing-Chi Cheung, The Hong Kong University of Science and Technology, China
- Daniel German, University of Victoria, Canada
- Lou Yiling, Fudan University, China
- Lingming Zhang, University of Illinois Urbana-Champaign, USA
- Song Wang, York University, Canada
- Lei Ma, The University of Tokyo, Japan
- Yutaro Kashiwa, NAIST, Japan
- Shinpei Hayashi, Tokyo Institute of Technology, Japan
- Weiyi Shang, University of Waterloo, Canada
- Zhen Ming (Jack) Jiang, York University, Canada
- Ahmed E. Hassan, Queen's University, Canada
- Yasutaka Kamei, Kyushu University, Japan

# 4 AIWare

# 5 Meeting Schedule

This seminar took place between March 25 to March 28, 2024. For the detailed agent, please refer to Table 1.

| | |
|---|---|
| **Day 0**: March 24, 2024 (Sunday) | |
| 15:00 - : | Check-in |
| 19:00 - 21:00: | Welcome banquet |
| **Day 1**: March 25, 2024 (Monday) | |
| 09:00-09:15 | Welcoming address - Zhen Ming (Jack) Jiang, Ahmed E. Hassan, Yasutaka Kamei |
| 09:15-09:45 | Ahmed E. Hassan "AIware in the Foundation Models Era" |
| 09:45-12:00 | Self introduction from each invitee |
| 12:00-13:40 | Lunch + Group Photo |
| 13:40-14:40 | Keynote - Lingming Zhang |
| | "Software Quality Assurance in the Era of Large Language Models" |
| 14:40-15:10 | Planning for discussion sessions |
| 15:10-15:30 | Coffee break |
| 15:30-17:30 | Breakout Sessions #1 |
| 17:30-18:00 | Free-time |
| 18:00-19:30 | Dinner |
| **Day 2**: March 26, 2024 (Tuesday) | |
| 09:00-09:15 | Introduction to the 2nd day Agenda |
| 09:15-9:30 | Breakout summary Report #1 |
| 9:30 - 9:45 | Breakout Sessions #2 |
| 9:45 - 10:15 | Coffee break |
| 10:15-12:00 | Breakout Sessions #2 |
| 12:00-13:00 | Lunch |
| 13:00 - 20:45 | Excursion and dinner |
| **Day 3**: March 27, 2024 (Wednesday) | |
| 09:00-09:15 | Introduction to the 3rd day Agenda |
| 09:15-10:15 | Breakout summary Report #2 |
| 10:15-10:45 | Coffee break |
| 10:45-12:00 | Breakout Sessions #3 |
| 12:00-14:30 | Lunch |
| 14:30-15:30 | Breakout Sessions #3 (Continued) |
| 15:30-17:30 | Breakout summary Report #3 |
| **Day 4**: March 28, 2024 (Thursday) | |
| 09:00-10:30 | Discussion Session Wrap-up |
| 10:15-10:45 | Coffee Break |
| 10:45-12:00 | Workshop conclusion and Looking Forward |

Table 1: Meeting Schedule for this Shonan Seminar

# 6 Discussion Topics

In this section, we will report a summary of our discussion outcomes for each of the discussed topics. For each topic, we will first summarize the current promising research and practice achievements. Then we will present some of the promising research and practice achievements. Finally, we will present our list of term research direction that we feel the field to be tackling.

## 6.1 Code Generation

The discussion on this topic is centered around "Towards trustworthy code generation in the context of real world software development".

### 6.1.1 Current Achievements

Code generation, which involves transforming natural language inputs into code, has seen significant advancements in research and practice. A variety of benchmarks and datasets, such as EvalPlus, CoderEval, and ClassEval, have been developed to evaluate and advance the field. Numerous industry tools and models, including ChatGPT, Copilot, CodeLlama, Bard/Gemini, CodeWhisperer, and proprietary Google tools, highlight the practical applications of code generation. Open-source large language models further contribute to this ecosystem. Beyond code generation, FMs are being leveraged for tasks such as test generation, enhancing trustworthiness by ensuring robust code outputs. Formal proof synthesis, powered by tools like GPT-f, LISA, Baldur, Proofster, DSP, and LeanDojo, also underscores the potential of LLMs to improve trustworthiness by generating formally verified code and proofs. Together, these advancements reflect a rapidly evolving landscape in automated code generation and verification.

### 6.1.2 Overlooked Challenges

Despite advancements in code generation, several challenges remain overlooked. The boundaries between different tasks categorized as code generation lack clarity, complicating the evaluation and standardization of methodologies. One challenge is managing varying levels of abstraction, where an LLM generates both an abstract model and corresponding code. While this allows for cross-verification, it risks giving users a false sense of confidence in the artifacts. Integrating reasoning capabilities into the LLM workflow is another challenge, as it requires designing systems that actively assist users while maintaining accuracy. Additionally, alternative representations of models could improve user comprehension and aid generation but remain underexplored. Finally, ensuring that LLMs are aware of their own limitations is crucial to prevent them from attempting tasks they are incapable of performing or answering questions outside their training. Addressing these challenges is essential for building more reliable and user-friendly code generation systems.

### 6.1.3 Future Research Directions

Future research directions in code generation emphasize creating a more cohesive ecosystem of tools and improving how users interact with generated models and specifications. A key focus is clarifying the interplay between multiple tools

in the tool chain, such as LLMs, static analyzers, and test generation frameworks. While these tools already exist, their integration and interoperability are not well-documented or shared. Another direction involves exploring how test generation and theory exploration tools can help users understand and refine generated models and specifications. From a human-computer interaction (HCI) perspective, presenting information in ways that align with user needs and ensuring generated outputs meet user expectations are critical areas for development. Moving up the abstraction stack is another priority, with promising research on FM-powered agent tree searches, etc. This approach could extend beyond proof generation to other coding tasks, paving the way for innovative applications. Finally, translating source code across programming languages offers additional opportunities for enhancing cross-language development and collaboration.

## 6.2 Software Productivity

The theme of the discussion on software producivity is around "Productivity with FMs as CoPilots".

### 6.2.1 Current Achievements

Software productivity in the context of LLM-based code generation tools refers to the efficiency and quality with which developers can achieve their goals, such as writing, reviewing, and debugging code. These tools have shown potential for improving task completion speed, aiding in code comprehension, and streamlining workflows. However, defining and measuring productivity remains complex, with debates around metrics like time saved, code quality, and developer satisfaction. Current research highlights challenges such as ensuring the quality of generated code, mitigating over-reliance on tools, and understanding their impact on team dynamics and skill retention. Despite these challenges, tools like Copilot and ChatGPT are recognized for enhancing developer tasks, though their long-term impact on productivity and workplace practices requires further investigation.

### 6.2.2 Overlooked Challenges

The challenges of defining and measuring software productivity with LLM-based tools remain largely overlooked. Productivity is often ambiguously defined, with metrics such as acceptance rate of suggestions, faster code completion, or improved code quality failing to capture the broader impacts on outcomes like bug-free or maintainable code. There is a need to shift the focus from "productivity" to "helpfulness" by assessing whether tools effectively aid developers in achieving tasks. Productivity also encompasses dimensions like developer satisfaction and happiness, balancing quality with efficiency, and understanding adoption impacts, as developers may feel pressured to use tools to keep up with peers. However, tools face inherent limitations, including generating incomplete or suboptimal code due to limited context and risks of over-reliance, which could diminish core coding skills over time. Economic and ethical concerns, such as job security and tool applicability across different organizational contexts, further complicate adoption. Measurement challenges include the lack

of tailored benchmarks for specific tasks and difficulty assigning business value to tool introduction. Additionally, social and collaborative dynamics, such as interactions with LLM-based tools or shifting documentation practices, influence how developers work. Longitudinal and domain-specific studies are needed to understand the long-term impacts on onboarding, team productivity, and development practices. Addressing these challenges will enable the research community to develop tools that align with meaningful productivity metrics while fostering quality and collaboration in software engineering.

### 6.2.3 Future Research Directions

Future research directions in software productivity with FM-powered software development focus on understanding how these tools impact developer workflows, learning curves, and overall efficiency. Key questions include whether AI-generated code accelerates the code review process and how exploratory interfaces (e.g., chat-based tools) compare with embedded tools like Copilot in integrated development environments (IDEs). Research should also investigate how these tools influence deep work, learning, and tinkering, particularly in inclusive contexts where diverse developer skills and backgrounds come into play. Pragmatic concerns include instrumenting platforms like GitHub to identify generated code, enabling the study of usage patterns, and understanding the economic trade-offs developers make when choosing tools. There is a growing need to clarify tool interfaces, ensuring they align with developer workflows rather than technology constraints, while also exploring how LLM-based tools lower barriers for beginners, aid with APIs, and streamline work in unfamiliar domains. Ethical and accountability challenges—such as responsibility for errors in LLM-generated code—remain critical areas for exploration. Additionally, ensuring equitable access to these tools and fostering inclusivity in research and development will be vital as the field evolves. In the long term, as these tools become permanent fixtures in development workflows, research must explore how to maximize their benefits across diverse tasks while mitigating potential risks. Intellectual property (IP) concerns and equitable access remain significant obstacles, drawing parallels with lessons learned from two decades of open-source software adoption, which fostered inclusivity and democratization. Ensuring equitable access to LLM-based tools is vital, especially for under-resourced communities and educational environments, where affordability may limit students' ability to leverage these tools beyond academia. Moreover, the integration of LLM tools in education raises critical questions about teaching methodologies, emphasizing the need to prepare students for a future where these tools are ubiquitous and ensuring their skills remain adaptable in a changing technological landscape.

## 6.3 Data Quality

The focus of this topic is on how to collect or generate high-quality data to train and test a FM.

### 6.3.1 Current Achievements

Data quality in software engineering encompasses a broad range of attributes that determine the utility, fairness, and reliability of data for various tasks. High-quality data is accurate, representative, timely, and diverse, avoiding excessive repetition while covering critical dimensions of a domain. The importance of specific data attributes varies depending on the context, the task, and the stage of model development—ranging from pre-training to fine-tuning and testing. For example, some quality issues might be acceptable in pre-training but are critical to address in fine-tuning or testing phases. Synthetic data has emerged as a controlled and scalable strategy for training, though it can pose challenges such as missing unseen data or risks of overfitting. Key advances include fine-tuning models with small, high-quality datasets, using curated repositories like GitHub for software-related data, and employing frameworks to assess and ensure data quality. Ongoing challenges involve automating quality checks, integrating diverse data types (e.g., runtime logs, documentation, diagrams), and addressing data contamination risks. Promising research efforts continue to explore these challenges, leveraging insights from curated software datasets and innovative data-centric approaches.

### 6.3.2 Overlooked Challenges

Overlooked challenges in data quality for software engineering include issues of data security, domain specificity, and maintaining dataset integrity. Poisoning training datasets at web scale, as demonstrated in recent research, highlights the vulnerability of large-scale data to malicious attacks that can compromise model outputs. Similarly, stealthy backdoor attacks targeting code models pose significant risks, emphasizing the need for robust defenses. Domain-specific models, such as those for repairing quantum programs, require highly tailored datasets, raising questions about the feasibility of obtaining sufficient, high-quality data and ensuring it remains current. Another critical area is the identification and management of software engineering-specific "data smells", such as inconsistencies, wrong labels, missing values, or incomplete information. Addressing these challenges demands both a deeper understanding of domain-specific requirements and the development of tools and strategies to safeguard, curate, and adapt datasets effectively.

### 6.3.3 Future Research Directions

Future research directions in software engineering (SE) and LLM include adapting established data quality dimensions—such as completeness, accuracy, timeliness, consistency, and accessibility—to the unique challenges of SE and LLM contexts. These efforts involve investigating issues like ensuring sufficient and reliable data breadth and depth, maintaining compatibility with historical data, and making information readily retrievable for practical use. Specific challenges include addressing known problems, such as the limitations of the SZZ algorithm, and exploring data attribution methods to trace and debug errors in training data, an intersection of SE and machine learning. Synthetic data generation also presents opportunities, particularly in designing methods to train models when linking disparate data sources is non-trivial. Emerging AI and LLM-driven techniques, such as reinforcement learning with human feedback

(RLHF) for data preprocessing and labeling, and automated tools for testing and profiling data quality, hold promise for improving data engineering practices. These approaches aim to make data management more efficient and robust, laying the groundwork for more reliable and high-performing AI/LLM-based systems. In the longer term, one should look into the critical issue of the creation of "training-free zones" on the internet to prevent contamination of training data, preserving the integrity of datasets. Convincing companies to share high-quality internal data for public use poses another significant challenge, requiring strategies to balance proprietary concerns with collective progress. Achieving the optimal trade-off between dataset size and quality remains a persistent question, as larger datasets often compromise on precision while higher-quality datasets can be harder to scale. Additionally, a lack of a "software process model" perspective, such as the V-model, limits the contextual understanding of SE datasets. Integrating and fusing SE data from diverse sources—spanning versions, modalities, and active or passive collection methods—presents another formidable challenge. Developing advanced reasoning methods, such as chain-of-thought reasoning, could help make sense of complex, multi-source datasets and enable better insights for SE and LLM applications. These challenges highlight the need for long-term innovation in dataset construction, usage, and quality assurance.

## 6.4 Agents

The discussion on this topic is centered around "FM-powered Agent-Oriented Software Engineering".

### 6.4.1 Current Achievements

Software engineering (SE) agents are autonomous systems that leverage the capabilities of foundation models (FMs) to achieve high-level goals in software development. Unlike traditional tools or APIs that require explicit, step-by-step instructions, SE agents operate independently by breaking down overarching objectives into manageable tasks and orchestrating their execution. These agents are designed to support software engineers by handling complex tasks, ranging from code generation and debugging to effort estimation and collaborative problem-solving.

Current achievements in this field include the development of intelligent interfaces like GitHub Copilot [7] and advanced agent frameworks such as MetaGPT [16] and Voyager [22]. These systems demonstrate the ability to assist in tasks like multi-agent collaboration, responsible AI design, and open-ended problem-solving. Additionally, efforts like modularized agents and communication optimization (e.g., language-based or visual interactions) are paving the way for more effective integration of agents into SE workflows. However, challenges such as maintaining trustworthiness, understanding causal relationships, and addressing the "black box" nature of agents remain critical for the future development of SE agents.

### 6.4.2 Overlooked Challenges

Overlooked challenges in the use of agents and LLM in SE highlight several fundamental gaps in understanding and implementation. One key issue is how agents should present themselves, other agents, and humans in a way that builds trustworthiness while ensuring their outputs are comprehensible and reliable. The complexity of causal relationships in agent systems and the difficulty in defining thresholds or safeguards to evaluate their effects further complicates their use. These systems remain prone to being "black boxes", which hinders transparency and trust. Additionally, applying autonomous techniques effectively within the SE context requires bridging gaps in skill measurement and communication methods between agents and human developers. Questions also arise about whether agent interactions will resemble human communication or API usage. Finally, the lack of proper scenarios, datasets, and benchmarks tailored for SE tasks, along with the need for industry collaboration to refine and safeguard multi-agent systems, remains a significant challenge that demands attention from both academia and industry.

### 6.4.3 Future Research Directions

Future research directions in the intersection of SE and FM-powered autonomous agents emphasize several critical areas. One major avenue involves exploring how agents can autonomously handle complex tasks by breaking down high-level goals into actionable subtasks and orchestrating their execution. Research could investigate the roles and communication methods of multi-agent systems, including how agents interact with each other and with humans, such as through languages, images, or other modalities. Trustworthiness and responsibility in agent systems are key challenges, requiring the development of safeguards, thresholds, and techniques for measuring agent skills and understanding their black-box nature. Practical applications, like GitHub bots and modularized agent systems, present opportunities for improving onboarding, training, and collaborative software development. Long-term challenges include creating benchmark datasets and scenarios specifically for SE contexts and addressing conflicts, infrastructure needs, and the complexities of maintaining versus generating code in agent-based systems. These directions promise to reshape SE practices and leverage agents for more efficient, trustworthy, and innovative workflows.

## 6.5 FM-powered Software Testing

The discussion on this topic span across the design, the execution, and the analysis of FM-powered software testing.

### 6.5.1 Current Achievements

Current achievements in FM-powered software testing demonstrate significant progress in utilizing LLMs to streamline and enhance traditional testing practices. LLMs have been successfully applied to both black-box testing, such as fuzz testing, and white-box testing, like unit test generation. They have proven effective in automating test case generation and repair, minimizing flaky tests, and reducing test suite duplication, making testing processes more efficient and

scalable. For example, tools like FuzzGPT [14] leverage LLMs to uncover runtime errors and explore metamorphic relations, which help generate better test cases. Moreover, LLMs are being used to map events from game documentation to generate test cases and to reproduce bugs from bug reports, demonstrating their versatility. Efforts to integrate LLMs with other techniques, such as retrieval-based approaches and differential prompting, further enhance their utility in producing high-quality tests. These advancements highlight the potential of LLMs to revolutionize software testing by automating labor-intensive tasks, improving coverage, and reducing costs, though challenges in oracle generation and hallucination remain areas for future exploration.

### 6.5.2 Overlooked Challenges

Overlooked challenges in FM-powered software testing stem from the complexities of effectively integrating large foundation models into diverse testing processes. One significant challenge lies in defining the boundaries of what these models can reliably achieve, particularly for generating accurate test cases and oracles. The lack of formal specifications for critical systems exacerbates this, as most rely on loosely defined requirements, making it difficult to ensure comprehensive testing coverage. Additionally, data security and the risk of backdoor attacks on training datasets threaten the reliability of models used for testing. FM-powered testing also faces scalability issues, such as optimizing test suite minimization and addressing flaky test cases, which are expensive and time-consuming to repair. Another challenge is ensuring the utility of FM-generated outputs, as hallucinations can mislead testing efforts without proper evaluation frameworks. Furthermore, the reliance on proprietary or commercial models raises concerns about accessibility, transparency, and trustworthiness in open testing environments. These challenges highlight the need for a deeper understanding of FM limitations, the development of robust evaluation methods, and advancements in data and testing frameworks to fully leverage FM capabilities in software testing.

### 6.5.3 Future Research Directions

Future research directions in FM-powered software testing focus on addressing critical challenges to enhance the reliability, efficiency, and adaptability of these approaches. A key area is leveraging FMs and metaphoric relations to generate test cases, particularly for complex and domain-specific programs. Generating accurate test oracles remains a significant challenge, requiring novel strategies to validate functional errors effectively. Another direction involves using FMs to identify inconsistencies and bugs within software implementations, such as comparing outputs derived from books and compilers. Automating the creation, maintenance, and evolution of testing processes through FMs is also critical, especially in multi-agent systems where human involvement must be seamlessly integrated. Additionally, incorporating richer feedback, such as system memory layouts or environmental states, could guide FMs to produce more effective and context-aware test cases. Addressing these challenges will require innovative methods to test in complex environments, capturing nuanced states and system-level inputs while ensuring that FM-based approaches align with broader software engineering goals.

### 6.5.4 IDE

This discussion of IDE was centred around the capabilities of the future IDE for AIWare.

### 6.5.5 Current Achievements

Current achievements in integrating LLMs into IDEs have significantly enhanced the software development process. Modern IDEs now include features like code completion and chat interfaces, enabling developers to interact dynamically with AI assistants. These assistants assist in generating code, identifying design patterns, and providing suggestions tailored to the context of ongoing development tasks. The integration of LLMs into IDEs has also allowed for real-time visualization of AI processes, making it easier for developers to understand the underlying mechanics of generated outputs. Additionally, IDEs are becoming more proactive, capable of adapting to changes in code and maintaining workflow continuity without unnecessary interruptions. These advancements have laid the foundation for prompt-driven development and domain-specific tooling, highlighting the growing role of AI-powered IDEs in streamlining complex software engineering tasks and improving developer productivity.

### 6.5.6 Overlooked Challenges

Overlooked challenges in integrating LLMs into modern IDEs stem from the complexities of adapting development environments to AI-driven workflows while maintaining developer productivity and ease of use. A major challenge lies in balancing traditional code-centric approaches with emerging prompt-centric paradigms, requiring new methods for managing and stabilizing prompts across model versions. IDEs need to evolve to handle diverse user needs, such as offering workflow-aware suggestions that adapt to the context, like whether the code will be reviewed by humans. Another critical gap is the lack of traceability between prompts and generated code, making debugging and maintaining AI-assisted outputs cumbersome. Additionally, the absence of a unified vocabulary for actions and elements within future IDEs complicates the automation of interactions and tasks. There is also a growing need to reconcile the rapid pace of industry adoption with academic research, ensuring that foundational challenges like consistency, error reduction, and seamless integration of testing and debugging workflows into AI-assisted environments are not sidelined. Addressing these issues will require collaborative efforts between software engineering and ML communities to create adaptable, proactive, and user-centric IDEs that fully leverage LLM capabilities.

### 6.5.7 Future Research Directions

Future research directions for integrating LLMs into IDEs revolve around redefining the interaction between developers and tools to enhance efficiency and innovation. A critical focus is creating IDEs that are context-aware, capable of adapting to user workflows, and proactive in offering design patterns and debugging assistance without intrusive interruptions. Research must explore transitioning from code-centric to prompt-centric development, addressing challenges

such as prompt stability across LLM versions and maintaining traceability between prompts and generated code. Long-term goals include leveraging LLMs for broader software lifecycle tasks like testing, debugging, architecture, and specification management. Collaboration with the machine learning (ML) community to design LLM architectures tailored for software engineering tasks is essential. Additionally, understanding how LLMs can autonomously identify requirements, propose fixes, and act as intelligent assistants or even product managers presents transformative possibilities. Collecting domain-specific data and designing traceable relationships between prompts and code are foundational steps for realizing this vision, paving the way for a new generation of developer tools.

## 6.6 Process

The theme of this discussion topic was software development processes in the Age of FM.

### 6.6.1 Current Achievements

Current achievements in software development processes in the era of FMs revolve around integrating FM-powered tools to transform traditional workflows. These processes now involve selecting models based on cost, privacy, and domain-specific requirements, with prompting, knowledge engineering, and context management becoming integral components. The use of dynamic and static workflows has expanded to accommodate FM capabilities, shifting artifact creation from manual coding to FM-assisted generation. Tools like Copilot and agentic systems such as Devika and Devin have redefined the nature of software artifacts and their generation. The ability to specify detailed requirements and ensure correctness and reliability in FM-driven environments has become a central focus, with early research showing promise in adapting FM-powered tools to streamline tasks like code analysis and defect detection. These advancements highlight a move toward FM-based building blocks that redefine traditional development while ensuring reliability and adaptability.

### 6.6.2 Overlooked Challenges

Overlooked challenges in the integration of FMs in software development processes highlight critical gaps in academic and industrial understanding. Key concerns include how to realign workflows to accommodate FM-driven development, particularly when tasks such as code generation scale exponentially, potentially overwhelming traditional practices like code review and analysis. There is uncertainty about how to adapt defect detection and debugging processes in FM-centric systems, especially when artifacts like requirements, code, and tests are inherently intertwined with probabilistic models. The transition from static to dynamic workflows introduces further complexity in ensuring correctness, reliability, and traceability. Questions around sustainability and regulation also remain underexplored, including the societal and economic implications of FM adoption, particularly in aligning with global regulations. Lastly, a critical gap lies in understanding the role of humans within FM-driven workflows and how

best to leverage human capabilities in tandem with AI tools, ensuring the development of robust, trustworthy systems. These challenges necessitate a rethinking of both technical and human-centered approaches to software engineering.

### 6.6.3 Future Research Directions

Future research directions in this topic focus on redefining processes to leverage FM-powered tools and autonomous agents effectively. Key areas include integrating FMs as core components in workflows, prompting dynamic and static workflow designs that adapt to domain-specific requirements. Ensuring correctness and reliability in FM-driven systems is critical, especially when FMs replace traditional databases or serve as dynamic interfaces to data. Requirements engineering will evolve to emphasize explicit specification and property guarantees for FM-driven interactions. Research must also address the scalability of traditional practices like code review and testing, adapting them to the higher code generation rates enabled by FMs. Additionally, the role of "AIware", the interplay between hardware, software, and FM agents—requires exploration to design cohesive systems. As development processes realign, sustainability and regulatory considerations will influence how FMs are deployed, ensuring human involvement in areas best suited to their capabilities. Long-term challenges include creating robust debugging processes, orchestrating complex dependencies among FMs, and fostering collaboration between academic and industry stakeholders for data and infrastructure sharing. These directions pave the way for a transformative shift in software engineering methodologies.

## 6.7 Education

### 6.7.1 Current Achievements

Current achievements in integrating FMs into software engineering education demonstrate significant promise in enhancing learning and teaching practices. Tools like GitHub Copilot have showcased practical applications in code generation, while research highlights their potential in areas such as automated scoring, creating exercises, and supporting students in problem-solving. Studies, including "Software Engineering Education Must Adapt and Evolve for an LLM Environment"[19] and "ChatGPT and Software Testing Education: Promises & Perils" [17], outline both the opportunities and challenges these tools present in adapting curricula to new technologies. LLMs have also been used to generate self-assessment quizzes, programming exercises, and code explanations, contributing to more interactive and accessible educational materials. Moreover, initiatives exploring the personalization of teaching content and gamification strategies highlight innovative approaches to leverage these tools for improved engagement. These advancements signal a transformative shift in software engineering education, driven by the integration of LLMs and their capabilities.

### 6.7.2 Overlooked Challenges

Overlooked challenges in the intersection of FM and SE education highlight the need for systemic adaptation to evolving technologies. A significant challenge is defining how to effectively teach software development and engineering

practices while integrating FMs like LLMs. This includes teaching students to distinguish good practices from anti-patterns and fostering a sense of "good taste" in software design, which remains difficult to formalize. Personalization of education through multi-agent systems or anonymization techniques (e.g., avatars) introduces complexities, including potential risks to data security and fairness. Another overlooked area is the shift in teaching methodologies, such as whether traditional programming and compiler design should still be taught in an era where FMs handle much of the coding. Additionally, the ethical implications and long-term impact of relying on FMs for coding education, as well as the need for a clear path from beginner to intermediate levels in programming with LLMs, demand further exploration. Addressing these challenges will require innovative approaches to redesigning courses, evaluating logical reasoning, and ensuring equitable access to FM-driven educational tools.

### 6.7.3 Future Research Directions

Future research directions in FM and LLM-powered software engineering education highlight the need to redefine teaching methodologies and integrate AI tools effectively into curricula. Key challenges include teaching foundational software engineering principles, such as distinguishing between good and bad practices, while leveraging LLMs to personalize education. Innovative approaches like gamification, multi-agent systems, and anonymized teaching avatars could help tailor learning experiences to individual needs. Research must also explore how to efficiently teach students to use LLMs across different implementations without becoming overly reliant on specific tools. As LLMs advance, questions arise about the necessity of teaching traditional programming skills versus emphasizing broader logical reasoning and problem-solving abilities. Additionally, evaluating the process of prompting and identifying logical reasoning in failures are critical for redesigning courses in this AI-driven era. Collaboration between academia and industry is essential to develop datasets and infrastructures to support these evolving educational paradigms. These efforts aim to prepare students for a future where FM and LLM tools are integral to software engineering.

## 6.8 Synergizing classical SE analysis and FM

The goal of this discussion is to find effective approaches/workflows to combine the strengths of the two worlds: classical software engineering analysis and the power of FM.

### 6.8.1 Current Achievements

Recent advancements in synergizing classical SE analysis and FMs have demonstrated promising capabilities in enhancing both fields. Achievements include the integration of static and dynamic SE analysis tools to generate context for feeding into LLMs, boosting their understanding and performance. Approaches like neuro-symbolic techniques combine information retrieval (IR) and LLMs to synthesize program snippets from user intents, addressing challenges in search and ranking. Sequential events from dynamic analysis have been adapted as context for LLMs, despite challenges with long sequences. Efforts to improve

FM utility through additional information, such as intra-project dependencies and retrieval-augmented generation (RAG), have also shown potential. Furthermore, FMs have been applied in classical SE tasks like probability-guided program analysis, enabling a balance between probabilistic insights and deterministic analysis methods. These advancements highlight the growing synergy between classical SE techniques and FMs, offering new pathways for innovation in program analysis, debugging, and feature transplantation.

### 6.8.2 Overlooked Challenges

Overlooked challenges in synergizing classical SE analysis with FMs include the need for better integration between traditional SE techniques and FM capabilities. A critical gap lies in leveraging non-LLM techniques, such as static program analysis (SPA), to provide stronger guarantees alongside FM-based approaches. Handling sequential events with dynamic analysis presents difficulties due to the length and complexity of sequences, requiring innovative methods to encode and process this information. Pretraining models with additional context, like project dependencies, and integrating more structured knowledge through techniques like retrieval-augmented generation (RAG) are underexplored. Another challenge involves enabling FMs to address program-specific behaviors by learning from traces, which could improve the prediction and understanding of software systems. Furthermore, creating feedback loops between FMs and SE tools, where analysis results are checked and iteratively refined, remains an ambitious yet crucial direction. Finally, automating complex tasks such as feature transplantation between programs, debugging misuse scenarios, and integrating tools like WhyLLM into the debugging workflow are pressing challenges requiring collaboration between academia and industry. These overlooked areas highlight the need for interdisciplinary approaches to fully realize the potential of combining classical SE and FM techniques.

### 6.8.3 Future Research Directions

Future research directions in synergizing classical SE analysis and FMs center around enhancing the capabilities of both approaches through deeper integration. Key areas include leveraging SE analysis tools, such as static and dynamic analysis, to provide enriched contexts for FM training and usage, ultimately boosting their accuracy and utility. Neuro-symbolic techniques, which combine information retrieval with FMs, aim to solve complex tasks like synthesizing program snippets from user intents, treating these challenges as search and ranking problems. Long-term goals involve LLM-guided program analysis and learning from program traces to better predict and understand program behaviors. Another promising direction is creating feedback loops between FMs and analysis tools, where outputs are validated and refined dynamically. Research also explores LLM-enabled feature transplantation, allowing automated adaptation of program features between projects, and the generation of realistic debugging and misuse scenarios. These directions highlight the potential for bidirectional enrichment: classical SE techniques enhancing FM capabilities and FMs revolutionizing traditional SE practices.

# 7 Conclusions

This meeting brought together world class researchers and practitioners to discuss challenges and opportunities in the intersection area between FMs and software engineering. Various topics ranging from code generation to software productivity are discussed. All the participants agree that, as we have entered into the generative AI era, the synergy between these two areas is just at the beginning and will continuously improve and evolve over time.

# References


[1] 1st acm international conference on ai-powered software (aiware). https://2024.aiwareconf.org/.

[2] Autogpt. https://github.com/Significant-Gravitas/AutoGPT.

[3] Chatgpt plugins. https://openai.com/blog/chatgpt-plugins.

[4] Fm+se vision 2030. https://fmse.io/vision/index.html.

[5] Prompt engineering guide. https://www.promptingguide.ai/.

[6] Ai language models are struggling to "get" math. https://spectrum.ieee.org/large-language-models-math, October 2022.

[7] Github copilot. https://github.com/features/copilot, 2022.

[8] A path towards autonomous machine intelligence. https://openreview.net/pdf?id=BZ5a1r-kVsf, 2022.

[9] Bloomberg intelligence: New report finds that the emerging industry could grow at a cagr of 42https://www.bloomberg.com/company/press/generative-ai-to-become-a-1-3-trillion-market-by-2032-research-finds/, June 2023.

[10] Bringing the world closer together with a foundational multimodal model for speech translation. https://ai.meta.com/blog/seamless-m4t/, 2023.

[11] Gartner places generative ai on the peak of inflated expectations on the 2023 hype cycle for emerging technologies. https://www.gartner.com/en/newsroom/press-releases/2023-08-16-gartner-places-generative-ai-on-the-peak-of-inflated-expectations-on-the-2 August 2024.

[12] L. Berglund, M. Tong, M. Kaufmann, M. Balesni, A. C. Stickland, T. Korbak, and O. Evans. The reversal curse: Llms trained on "a is b" fail to learn "b is a", 2024.

[13] R. Bommasani, D. A. Hudson, E. Adeli, R. B. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. S. Chatterji, A. S. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. E. Gillespie, K. Goel, N. D. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Koh, M. S. Krass, R. Krishna, R. Kuditipudi, and et al. On the opportunities and risks of foundation models. *CoRR*, abs/2108.07258, 2021.

[14] Y. Deng, C. S. Xia, C. Yang, S. D. Zhang, S. Yang, and L. Zhang. Large language models are edge-case fuzzers: Testing deep learning libraries via fuzzgpt, 2023.

[15] H. Guo, J. Yang, J. Liu, L. Yang, L. Chai, J. Bai, J. Peng, X. Hu, C. Chen, D. Zhang, X. Shi, T. Zheng, L. Zheng, B. Zhang, K. Xu, and Z. Li. Owl: A large language model for it operations, 2024.

[16] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, C. Zhang, J. Wang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2024.

[17] S. Jalil, S. Rafi, T. D. LaToza, K. Moran, and W. Lam. Chatgpt and software testing education: Promises amp; perils. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, April 2023.

[18] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12), Mar. 2023.

[19] V. D. Kirova, C. S. Ku, J. R. Laracy, and T. J. Marlowe. 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*, SIGCSE 2024, page 666–672, New York, NY, USA, 2024. Association for Computing Machinery.

[20] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools, 2023.

[21] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face, 2023.

[22] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023.

[23] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus. Emergent abilities of large language models, 2022.

[24] S. Wu, O. Irsoy, S. Lu, V. Dabravolski, M. Dredze, S. Gehrmann, P. Kambadur, D. Rosenberg, and G. Mann. Bloomberggpt: A large language model for finance, 2023.