Software Engineering Education Must Adapt and Evolve for an LLM (Large Language Model) Environment

Vassilka D. Kirova
Bell Labs Consulting, Nokia, New Providence NJ, USA
+1 973-517-3410
vassilka.kirova@bell-labs-consulting.com

Cyril S. Ku
William Paterson University, Wayne NJ, USA
+1 973-720-2952
kuc@wpunj.edu

Joseph R. Laracy
Seton Hall University, South Orange NJ, USA
+1 973-275-2081
joseph.laracy@shu.edu

Thomas J. Marlowe
Seton Hall University, South Orange NJ, USA
+1 732-381-3372
thomas.marlowe@shu.edu

ABSTRACT

In the era of artificial intelligence (AI), generative AI, and Large Language Models (LLMs) in particular, have become increasingly significant in various sectors. LLMs such as GPT expand their applications, from content creation to advanced code completion. They offer unmatched opportunities but pose unique challenges to the software engineering domain. This paper discusses the necessity and urgency for software engineering education to adapt and evolve to prepare software engineers for the emerging LLM environment. While existing literature and social media have investigated AI's integration into various educational spheres, there is a conspicuous gap in examining the specifics of LLMs' implications for software engineering education. We explore the goals of software engineering education, and changes to software engineering, software engineering education, course pedagogy, and ethics. We argue that a holistic approach is needed, combining technical skills, ethical awareness, and adaptable learning strategies. This paper seeks to contribute to the ongoing conversation about the future of software engineering education, emphasizing the importance of adapting and evolving to remain in sync with rapid advancements in AI and LLMs. It is hoped that this exploration will provide valuable insights for educators, curriculum developers, and policymakers in software engineering.

CCS Concepts

Software engineering→Software creation and management.
Social and professional topics→Professional topics→ Computing education→Computing education programs→ Software engineering education.
Artificial intelligence→Natural language processing→Natural language generation.
Social and professional topics→ Professional topics→Computing profession→Code of ethics.

Keywords: Software engineering; software engineering education; generative AI; large language models (LLMs); responsible AI; ChatGPT; software ethics; software engineering ethics

ACM Reference format:
Vassilka D Kirova, Cyril S. Ku, Joseph R. Laracy, Thomas J. Marlowe. 2024. Software Engineering Education Must Adapt and Evolve for an LLM (Large Language Model) Environment. In *Proceedings of 2024 ACM Technical Symposium on Computer Science Education (SIGCSE'24), March 20–23, 2024, Portland, OR, USA*, ACM, New York, NY, USA. 7 pages. https://doi.org/10.1145/3626252.3630927

SIGCSE 2024, March 20–23, 2024, Portland, OR, USA

ACM 979-8-4007-0423-9/24/03…$15.00
https://doi.org/10.1145/3626252.3630927

1 INTRODUCTION AND RELATED WORK

The advent of artificial intelligence (AI) has brought transformational changes across numerous sectors, encompassing healthcare, finance, entertainment, and in our domain of interest in this paper, education. Foremost among the AI revolution is the inception and ascendance of Large Language Models (LLMs) which possess the unique capability to generate human-like text. Their rapidly expanding applications present both unparalleled opportunities and unique challenges, including to the software engineering landscape.

Software engineering education is at a crucial juncture due to this AI-driven technology. LLMs like ChatGPT from OpenAI and BERT by Google, exemplify the capabilities of modern AI, finding applications in content creation, chatbots, and advanced code completion tools. With the escalating influence of LLMs, there is an urgent need to reshape the education of future software engineers, ensuring they can adeptly maneuver, contribute to, and ethically manage this LLM environment. Modern software engineering curricula need to holistically embrace the myriad facets of LLMs, ensuring students are prepared for the challenges and opportunities they present.

A wealth of literature has been dedicated to the integration of AI, particularly in areas such as Software Engineering [17]. There is

literature concerning the impact of ChatGPT on education [33]. Educators have explored ways to incorporate AI into various educational disciplines [34], with computer science and software engineering curricula being no exception [1, 3]. The rapidly evolving nature of AI presents a unique challenge for curriculum design and pedagogy. The integration of AI principles, machine learning techniques, and the management of AI's ethical and societal implications into software engineering education remains an area of active discussion. However, the specific focus on LLMs and their implications for software engineering education remains relatively unexplored.

The rise of LLMs has been marked by significant research, with studies focusing on capabilities, strengths, and limitations [11, 37]. It is broadly recognized that these models, while powerful, are far from perfect, with notable concerns surrounding their susceptibility to biases, misuse, and their "black box" nature [8]. These challenges are not isolated to the realm of AI research, but permeate the practical application of these models, making them highly relevant for software engineers.

The burgeoning domain of AI ethics has also garnered significant attention in academia. This is mirrored in the literature, with scholars highlighting the importance of understanding and addressing potential biases, transparency, privacy issues, and ethical considerations in AI [23, 35]. Importantly, these ethical considerations are not solely impacting the domain of AI researchers; they are pertinent to software engineers tasked with developing and deploying AI-powered applications, further underlining the necessity for ethics education in software engineering curricula.

This paper explores how software engineering education must adapt to the rising prominence of LLMs, reflecting the need for a holistic understanding of these models. We delve into several pivotal areas of knowledge for future software engineers, including a technical understanding of LLMs and AI principles, the ethical implications of AI deployment, data science techniques, debugging and interpreting AI model outputs, adaptability in a rapidly changing field, model training and tuning, the integration and application of LLMs in software systems, understanding relevant laws and regulations, and the pedagogical approaches most effective for teaching these complex topics.

By exploring these crucial areas of software engineering education in an LLM environment, we aim to shed light on the competencies that future software engineers will need. It is our hope that this discussion will contribute to the ongoing dialogue surrounding the future of software engineering education and serve as a valuable resource for educators, curriculum developers, and policymakers in the field. We also examine the question of how ChatGPT and similar tools and environments can be used effectively in the education process, specifically in software engineering education.

The rest of this paper is organized as follows: Section 2 considers the goals of software engineering education in light of student curriculum and prospective careers. Section 3 then examines the changes to software engineering, software engineering education, and course pedagogy. Section 4 focuses specifically on ethical issues. Finally, Section 5 presents conclusions and future work.

2 GOALS OF SOFTWARE ENGINEERING EDUCATION

To fully evaluate the impact of generative AI/interactive LLMs on software engineering education, we first consider the goals of software engineering education. These will be affected to some extent by context–the degree (or certificate) program in which it occurs, and the placement in that program. For simplicity, we assume a single framework: an upper-level sequence in a relatively large computer science or informatics program, perhaps in a software engineering track, and possibly serving as a capstone experience, although elements of the software development process and practices will have been covered in lower-level courses. The following recommendations may not be suitable in their entirety for every program, depending on its size, structure, and nature, and should be selectively adopted and modified accordingly. Moreover, these recommendations are intended to complement rather than replace ACM/IEEE guidelines [26, 29, 30] for courses in software engineering.

Such a software engineering sequence has a four-fold objective: (1) to convey to the student the conceptual foundations of software engineering, (2) to develop or strengthen student knowledge, skills, and capabilities in essential aspects of software engineering, (3) to prepare students for careers in software development or related fields, and to inculcate or refine student soft skills and capacity for life-long learning, and (4) to place software engineering in context within the program and within the discipline of computer science, and within the student's overall education.

Conceptual and structural foundations aim to provide an understanding of the software engineering life cycle, from ideation and requirements to maintenance [9, 36, 39, 42]. It should, after a brief historical overview, not merely comprise discussion of software engineering workflows, of the motivation behind modern approaches such as lean and agile [6, 15], and of software development as part of broader system and enterprise structures. It should also support foundational consideration of issues such as (1) software architecture, both the linkage of the software application/module into the larger system, and important principles and building blocks such as modularization, separation of concerns, patterns [19] (with data analysis and AI-driven patterns likely to emerge), aspect-oriented paradigm, and use of the functional paradigm; (2) requirements–elicitation, analysis, and documentation–including regulatory and legal requirements, structural and physical constraints and connections, user- and service-oriented constraints and goals, quality objectives, and methods and practices such as hypothesis-driven development (HDD) [13], behavior-driven and test-driven (BDD, and TDD) development [38], and design thinking [31]; (3) testing, validation, and verification (TVV), including formal methods; and (4) special attention to critical non-functional attributes such as security, privacy, safety, reliability, availability, scalability, timeliness and embedded systems behavior, and accessibility [36].

The second goal is to develop professional competence. The sequence should build upon and interleave with the conceptual view to introduce modern software engineering practice, incorporating hands-on experience and a major team project as

practicable. Ideally, the project (or projects) should be developed in collaboration with real-world partners, on problems related to their own practice or needs (e.g., capstone projects), where the partners serve as customer representatives, and instructors as product owners (as in agile) or technical manager–or in some circumstances, vice-versa. Both discussion and project should be based on up-to-date usage and course objectives to the extent possible–although gradual development of student knowledge, skills and competencies will almost inevitably make the project more waterfall-like than in actual practice. Students should, to the extent possible, make use of tools such as software configuration management, tool suites implementing UML [27], testing tools, and program analyzers. They should practice an Agile approach (for example, Scrum [40]) for planning, work management, team coordination and meetings, and tracking progress [15]. They can also use agile project management tools, such as Atlassian Jira™ [25]. The project should go through several iterations, a minimum viable product (MVP) delivery, and a release. The relationship of development activities to project management, systems engineering goals, product line, risk management, and corporate policy, again tied to the actual project(s) as far as possible, needs to be discussed.

Third, the course should aim to prepare the student for a career in software engineering, technical management, or a related field, and for life-long learning. Beyond professional competence, there are three major subgoals: to develop critical thinking and modeling habits and skills, via open-ended but focused inquiry; to refine the ability to acquire and integrate knowledge, such as through preparation of an annotated bibliography and brief overview; and to sharpen and specialize soft skills, such as communication (speaking and listening, writing and reading, and presentation), teamwork (leadership, support, respectful disagreement, and collaboration), documentation-related tasks, and professional interpersonal interactions. An interdisciplinary perspective is also desirable, but is more likely to be emergent than a specified course goal.

Finally, for full understanding, software engineering has to be placed in context. The greater systems context has already been mentioned, but other connections are likewise important. Software engineering should be placed within the degree curriculum. Connections to other computer science courses (such as programming and Web development, architecture and operating systems, data structures and algorithms, databases, networking, or security) are fairly straightforward, as are those to mathematics and data science courses (such as discrete structures, logic, or statistics), but those connections should not be left to the student. The use of generative AI will render a prerequisite course in data science (or in artificial intelligence), introducing data mining, machine learning, and neural networks, highly desirable. But other connections are also important and should be mentioned: ethics (see Section 4), communication and the social sciences (for user interaction, data visualization, interface design, cognitive science, and more), economics and business, law (and politics), and application areas such as the digital humanities or libraries. A historical understanding of the interaction of technology and society is also desirable, but arguably can be covered in a separate course.

3 SOFTWARE ENGINEERING EDUCATION IN AN LLM WORLD

The use of generative AI, and interactive LLM models in particular, in the software development process and the practice of software engineering raises three major questions: How does it affect the process and practice? How does it affect what we teach? How does it affect how we teach?

3.1 Changes to real-world practice

It is now clear that generative AI can influence all aspects of the software engineering process and all workflows in the development process, to be discussed further in our companion paper [32] (see also [7, 16]). Its LLM databases and fluency with language can ease requirements acquisition, refinement, and analysis, specification, and documentation, although it will work best if those databases prioritize relevant context and glossaries, and include examples of various process and project artifacts (such as various UML models [10]). It can also serve as a coder for routine tasks, and as a pair programming partner (GitHub) for more complex or more sensitive modules, as well as identifying opportunities for patterns and the need for refactoring or further documentation. It can also take rough sketches of narratives, codes, or artifacts, flesh them out, and render them in a consistent style. And it can also be used as an adjunct in technical project management, to look for missing elements or additional analyses. As such, considering an incremental iterative process (such as agile or lean software engineering), it is likely to increase efficiency and effectiveness and shorten the time between releases, allow more to be managed in each iteration, or facilitate more effective collaboration, as well as to change the granularity, scheduling, interleaving, and interaction of workflows and activities.

The use of interactive LLM models does however pose three significant challenges. One comes from the well-documented problem of fabulation and hallucination—citations and events in narratives are not trustworthy, and need to be checked. In addition, as with other AI applications, bias may have been embedded in the dataset, and emerged, particularly in requirements or user experience/interface workflows [4, 20, 21, 44]. A second arises because, no matter how complete or how specialized and contextual the databases and training sets are, and no matter how well prompts and interaction proceed, the tool does not *understand* the larger context, and may conflate meanings, or miss problems involving critical attributes such as security or privacy. And the third is the issue of intellectual property [12], with two dimensions: inflowing, where an interactive LLM tool may "plagiarize" code or narrative that is the property of others or covered by an open software arrangement, endangering the rights of the developer or the customer, and outflowing, where exchanges with the tool or artifacts produced by the tool may become part of its ongoing datasets, and leak proprietary artifacts or information to others.

Because of these challenges, some have suggested [43] that interactive LLM tools not be used in developing commercial

applications, but in our view, economic pressures, competition, and changes in quality standards will result in their general use. Still, three safeguards suggest themselves. The first is the use of an in-house instance of the tool, with properly filtered, seeded, and reviewed databases. The second is a more rigorous and thorough use of review and testing, validation, and verification (TVV), as further discussed below. And third is mostly in the future: greater integration with interactive LLM of the results of TVV activities, of data analytics, and of program analyses and analyses of other process artifacts.

3.2 Changes to course content

An upper-level software engineering course must make selective use of the changes described above, and as a result, other changes, both global/structural and focused, must occur. The evident global changes fall into three areas, closely following the points in the previous section.

First, as interactive LLM tools take over or assist with routine tasks, there will be a shift from practicing and refining detail to a more conceptual understanding, analogous in some ways to the shift in arithmetic with the advent of calculators, or the shift in statistics with spreadsheets and statistical software. With the time saved, the course can further explore key topics, including design thinking for innovation, software engineering workflows and artifacts, the nature and form of requirements and specifications, the use of patterns, aspects, functional language features, and refactoring [18], coding practices including clean code, code consistency, useful commenting, TVV, maintenance, and evolution. Most importantly, time should be spent discussing or reviewing the impacts of AI and the nature and use of generative AI and LLM models.

Second, since the process will be relying on interactive LLM tools, time needs to be allocated to preparing and interacting with those models. While it may not be reasonable to have a specialized training set or database, the course should consider what additional information, glossaries, and interactions would be helpful in preparing to use the LLM model to assist in the current project. More importantly, developing good prompts and sequences of interactions and refinements, and providing requests/suggestions based on review, analyses, and TVV activities, now becomes its own topic in an LLM-mediated software engineering program. Further, software engineering students should get familiar with the APIs and options for interaction provided by LLM-based tools.

Third, especially in the course project, far more time and attention can and must be devoted to review, testing, validation, and verification activities (R-TVV). This is required in part due to fabulation and hallucination, on the one hand, and context/semantic confusion, on the other. But it is also true that AI-based tools cannot (at least at this point) have a human understanding of issues such as safety, privacy, and physical reality, and that LLM models have a known problem with even simple proofs [5]. Partly for this reason, more time should also be devoted to maintenance and evolution tasks, ideally complemented by at least two iterations of the course project, using generative AI to aid in identifying possible improvements.

This may be the most important change, and deserves some elaboration. The key goals for software development have been characterized as "Do the right thing," "Do the thing right," and "Do the thing well," to which one might add "And don't screw up any guarantees."

- Review should include a requirements review for the fidelity of translation and integration of sources, consistency, coherence, completeness, and lack of ambiguity (while allowing for emergent requirements as in agile approaches), consistency checks between workflows (e.g., requirements to specification, specification to design, design to implementation, implementation to deployment), and consistency between artifacts and documentation.
- Testing, as is well understood, includes not only unit and integration testing, but platform and acceptance testing, and testing for security, safety, accessibility, and other critical guarantees.
- Validation checks whether the application being developed actually does what was envisioned, and whether results of the desired type(s) can readily be obtained by the user on the deployment platform.
- Finally, verification complements testing by providing assurances that the software performs as specified. Verification uses formal methods in various forms, such as protocol checking via temporal logic, model checking, static analysis, and hybrids, and can be combined with testing in dynamic analysis of specific scenarios. Verification can provide guarantees that may be hard to achieve by testing; for example, to check for memory leaks, access control failures, or timeliness for real-time and embedded applications. It should however be noted that static analyses and other formal approaches can be subject to false positives–detecting problems where there are none–often requiring a sequence of analyses.

While there will likely not be time to cover all of these R-TVV activities in depth—a good topic for an advanced undergraduate seminar—students should be exposed to these concepts, see them in operation, and be asked to implement at least a reasonable selection across the categories.

Focused changes to specific workflows and activities, beyond the further emphasis on review and TVV activities include the use of interactive LLM in (1) requirements acquisition and documentation–properties of and issues with similar tools, and capturing user dissatisfaction, expectations and desires; and (2) user experience/interface design and refinement, possibly in combination with a discussion of hypothesis-driven design.

3.3 Changes to course pedagogy

The use of generative AI, and of interactive LLM models in particular, introduces both opportunities and difficulties for instruction, both in general and for software engineering in particular. The curriculum-wide difficulty with students having ChatGPT or similar tools doing their assignments, and the concomitant problems with the tool's (inadvertent) plagiarism, and with fabulation and hallucination, and with students so reliant on such tools that they fail to understand or internalize key course

concepts, has been discussed at length [5, 41]. But such tools also provide an opportunity, used as intermediaries for improved English-language communication for students with disabilities and English as a second language speakers. They can likewise be used more generally to improve drawings, technical artifacts, and formatting.

In software engineering and in other courses, the difficulties can in part be addressed by giving open-ended assignments where (at least at present) interactive LLM tools usually do not have sufficient context or training to readily provide useful answers in complex scenarios. A favorite assignment of one of the authors is to give a partial description of a problem, and to ask: Do you know enough to solve the problem? If not [which is the correct answer]: What else do you need to know? Why does it matter? How will you make use of the information?

Similarly, students can be asked to locate and address risks from fabulation/hallucination using a combination of critical and reflective thinking with cross-checking. Such critical thinking skills are also important both in review, as discussed above, and in formal verification. An annotated bibliography assignment, even if ChatGPT or other tools can be used, can be kept honest by requiring this cross-checking, by asking for the student's evaluation of the paper, and by requiring a brief oral survey presentation.

LLM tools can also support interdisciplinary perspectives and collaborations–one possibility is simply to ask the tool to rewrite a document from a different perspective—both allowing more effective coverage of interdisciplinary links, as discussed in Section 2, and enabling cross-disciplinary applications. By suggesting data structures, patterns, refactoring, and other features, and analogs in other workflows, the tools can also improve the quality of the final application. Likewise, LLM tools and the time saved by their use may allow (and provide time for) better integration of results from data analytics and other AI techniques, where appropriate.

Team interactions can benefit, as mentioned above, from style consistency afforded by LLM tools, and by reduced need for team members to generate full documentation. The LLM tools in combination with other tools can also assist in project management, including risk management, although clearly not at a professional/commercial level, which would require additional activities, checks and validation.

The big remaining problem is one of assessment. Some suggestions for individual assignments—are provided above, but the usual difficulty of grading team projects has if anything been made more difficult. One possibility might be to require copies of original draft artifacts and prompts along with the finished product, and/or a critical evaluation of the result. Another might be, analogous to one of the assignments just mentioned, to require an assessment of strengths, weaknesses, and possible future directions for the application. Overall, it seems to be possible to offer an LLM-intensive software engineering course/sequence, by focusing the course on concepts as well as critical and reflective thinking, and on creating habits of mind for professional and personal success.

4 ETHICS AND RESPONSIBLE AI

The impact of AI on society is immense and multi-faceted, and LLMs are a prime example. Responsible AI encompasses a broad range of concerns, including bias, transparency, privacy, society impact, legal implications, and protection of intellectual property. Sound ethical reasoning must guide the development and deployment of AI. Professional scientists and engineers cannot remain at the emotivist level, i.e., attempting to make ethical decisions based on feelings, which are often unreliable and fleeting. Ethics, sometimes referred to as "moral philosophy," is a rigorous academic discipline that studies acts of the will affirming or rejecting the order proposed by reason.

The teleological, deontological, and virtue approaches to engineering ethics are all applicable to AI. Teleological ethics focuses on the *telos*, i.e., end goal of an act. Thus, teleological approaches to ethics judge an act to be right or wrong in relation to the outcome, i.e., I deliberately order my actions toward the realization of an outcome which I previously identified as good. Deontological approaches are rooted in *deon*, i.e., duty, and thus judge the ethics of an act based on the inherent rightness or wrongness, rather than on its consequences. Virtue ethics is based on *virtus*, i.e., excellence of character, and can be classified as a species of teleological ethics. It is established on the principle that one should "act in a way the good or virtuous person would act in the circumstance" [24]. As Anderson et al. [2] point out, Catholic Social Teaching (CST), or its analogs from other traditions, can also be fruitfully applied to ethical issues that arise in engineering and technology.

All students in engineering disciplines, including but not limited to computer science, electrical and computer engineering, and data science, should have a dedicated course in professional ethics. Such courses typically address ethical issues in the practice of engineering such as safety and liability, professional responsibility to clients and employers, whistle-blowing, codes of ethics, and legal obligations. They also provide philosophical analysis of normative ethical theories and review numerous case studies. Given the issues that are arising with AI around fairness, accountability, transparency, and privacy, a substantial number of classroom hours should be set aside for AI. For example, bias in AI, often resulting from biased training data, may lead to skewed outcomes and unjust discrimination. This can have very serious implications, particularly when AI is used in sensitive areas such as hiring, lending, criminal justice, or law enforcement.

Transparency, another key ethical principle, relates to the ability to understand and interpret how AI models make decisions. However, as mentioned above, LLMs often function as "black boxes," with their decision-making processes obscured and difficult to understand. This lack of transparency can complicate efforts to ensure the accountability of AI systems. LLMs can potentially reveal sensitive information, i.e., privacy, especially if they were trained on private data. Safeguarding user privacy requires careful consideration when developing and using AI technologies. LLMs can also generate original content, be it an article, a piece of music, an image, or software code, raising questions about authorship and ownership. The question of whether AI can hold a copyright or patent appears resolved in the negative for now, but may yet

change. In addition, the use of copyrighted materials in training machine learning models can also pose legal challenges.

The well-known problem of hallucination and fabulation in LLMs raises additional concerns for veracity, transparency, and explainability. In some cases, consequences may even impinge on safety, security, privacy, and more. This issue may be more evident in software development, since there is a need to provide greater documentation for the project and its evolution, for users, and others. There is thus an ethical as well as a practical obligation to review and verify artifacts generated by LLMs, and in particular, citations and accounts referenced in those artifacts, not only for the enterprise, and its clients and end-users, but also to the discipline, regulators, and the scholarly community. Of course, generative AI may also be able to support exploration, subject to supervision and/or review, of ethical risks, in combination (as discussed above) with various tools and analyses, to search for risks and potential ethical violations, and if readily available, such analyses should definitely be explored within the educational context and in actual application development in the real world.

Various public and private organizations have already become very active in this space. Software engineering students should be introduced to their contributions. A university course that addresses AI ethics should expose students to the work thus far by the ACM, IEEE, EU, UNESCO, Google, IBM, and others. A potential assignment might be to compare, contrast, and ultimately evaluate their approaches. Complementarily, ethics should be revisited within the software engineering course/sequence, with a focus on ethical development, deployment, evolution, and impact on critical concerns such as security, safety, and privacy, and these concerns should specifically be included in the course project, especially in requirements analysis and reviews.

5 CONCLUSIONS

In this paper, we argue that generative AI and LLMs, such as OpenAI ChatGPT, GitHub Copilot, and Google LaMDA will have a significant impact on the discipline and practice of software engineering and that software engineering education needs to evolve and adapt continuously to reflect these new developments and upcoming improvements to better prepare software engineering students for a successful career in software development, analysis, management, and related fields.

In the paper we examine the essential educational goals of software engineering curricula and then discuss how generative AI and LLMs impact, support, and further these goals. Generative AI and LLMs will change the way software engineers learn and work. They will change what software engineers do and how they do it. For this, software engineering educators will need to update the content of software engineering courses, including topics, tools, projects, and exercises. In summary, we see the following key areas of change:

- Software engineering curricula will need to incorporate focused discussions of the capabilities and limitations of LLMs, including how to use them effectively, how to interact and interface with them, including prompt engineering and APIs, how to evaluate their outputs, and how to deal with possible ethical and legal questions related to using them.
- The coverage of software engineering development and management workflows must be updated to reflect and explain the potential role of generative AI and LLMs.
- Software engineering curricula will need to further emphasize conceptual aspects, modeling, design thinking, problem-solving, HDD, and creativity, as LLMs can automate or simplify some of the low-level coding tasks.
- Software engineering curricula will need to incorporate more interdisciplinary and collaborative skills development, as LLMs can enable new applications and domains that require knowledge from different fields.
- Software engineering curricula will need to foster critical and reflective thinking, as LLMs can hallucinate and generate incorrect, incomplete, inconsistent, or biased outputs that may have negative consequences.

Further, as discussed in the paper, educators need to update pedagogical methods and tools, to allow for supervised and effective use of tools such as ChatGPT in the classrooms and as part of students' individual learning process. While the initial reaction of the education community to the introduction of ChatGPT was largely negative, this has been changing steadily [14, 22].

We see these tools as effective assistants in the education process, and we are incorporating them into our teaching and research plans. We are currently working on a detailed analysis of the impact of generative AI and LLMs on software engineering workflows including aspects as project and risk management. We also intend to look at questions as:

- How to ensure the quality, reliability, security, and ethics of generative AI outputs and systems, especially in safety-critical or high-stakes domains?
- How to leverage generative AI to support collaborative and interdisciplinary software development projects, involving diverse stakeholders, domains, and data sources (as a natural extension of our earlier work on collaborative software development [28])
- How to foster a culture of innovation and creativity among software engineering students and educators, using generative AI as a source of inspiration, feedback, or exploration?

Our interests and focus are on approaches that can be applied in practice, as part of a hands-on educational process and also by practicing software engineers.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Kathy Herbert of Montclair State University, Dr. Yong-Fong Lee of Intel, Mr. Garett Chang of Highstep Technologies, and Dr. Nagib Callaos of IIIS for valuable discussions on this topic.

REFERENCES

[1] A. Ahmad et al. 2023. Towards Human-Bot Collaborative Software Architecting with ChatGPT, EASE '23: *Proc. 27th International Conference on Evaluation and Assessment in Software Engineering*, June 2023, Pages 279–285. https://doi.org/10.1145/3593434.3593468

[2] J.M. Anderson et al. 2020. A Multi-Disciplinary Analysis of Catholic Social Teaching with Implications for Engineering and Technology. *J. Systemics, Cybernetics and Informatics* 18 (6), 41–49, 2020. http://www.iiisci.org/journal/sci/issue.asp?is=ISS2006

[3] P. Banerjee et al. 2023. Understanding ChatGPT: Impact Analysis and Path Forward for Teaching Computer Science and Engineering. Preprint, ResearchGate. April 2023. DOI: 10.36227/techrxiv.22639705.v1. License CC BY 4.0. https://www.researchgate.net/publication/370148926_Understanding_ChatGPT_Impact_Analysis_and_Path_Forward_for_Teaching_Computer_Science_and_Engineering

[4] Y. Bang et al. 2023. A multitask, multilingual, multimodal evaluation of chatgpt on reasoning, hallucination, and interactivity, arXiv preprint arXiv:2302.04023.

[5] M.Bartels. 2023, You Can Probably Beat ChatGPT at These Math Brainteasers. *Scientific American* (May 25, 2023). https://www.scientificamerican.com/article/you-can-probably-beat-chatgpt-at-these-math-brainteasers-heres-why/

[6] K. Beck et al. 2001. Manifesto for Agile Software Development. agilemanifesto.org

[7] B.A. Becker et al. 2023. Programming Is Hard - Or at Least It Used to Be: Educational Opportunities and Challenges of AI Code Generation. In *Proc. 54th ACM Tech.l Symposium on Comp. Sci. Educ., SIGCSE 2023*, Vol. 1. Association for Computing Machinery, New York, NY, USA, 500–506. https://doi.org/10.1145/3545945.3569759

[8] E.M. Bender et al. 2021. On the Dangers of Stochastic Parrots: Can Language Models be too Big? In *Proc. 2021 ACM Conference on Fairness, Accountability, and Transparency, FAccT'21*, March 3-10, 2021, Canada, Pages 610–623. https://doi.org/10.1145/3442188.3445922

[9] E.J. Braude and M.E. Bernstein. 2016. *Software Engineering: Modern Approaches* (2nd ed.). Waveland Press, Inc., 2016.

[10] G. Booch et al. 1999. *The UML Users Guide*, Addison-Wesley, Reading, MA, 1999.

[11] T.B. Brown et al. 2000. Language models are few-shot learners. arXiv preprint arXiv:2005.14165.

[12] R.D. Caballar. 2022. Ownership of AI-Generated Code Hotly Disputed: A copyright storm may be brewing for GitHub Copilot, *IEEE Spectrum*. https://spectrum.ieee.org/ai-code-generation-ownership

[13] A. Cho. Hypothesis-Driven Development. https://www.ibm.com/garage/method/practices/learn/practice_hypothesis_driven_development/

[14] M. Daun and J. Brings. 2023. How Will ChatGPT Change Software Engineering Education? In *Proc. 2023 Conf. on Innovation and Tech. in Comp. Sci. Education, ITiCSE 2023*, Vol. 1. June 2023. 110–116. https://doi.org/10.1145/3587102.3588815

[15] Digital.ai, "16th State of Agile Report", 2022, last accessed 12/10/2023.

[16] C. Ebert and P. Lourida. 2023, Generative AI for Software Practitioners, *IEEE Software* 40 (4), 2023, 30–38.

[17] R. Feldt et al. 2018. Ways of Applying Artificial Intelligence in Software Engineering, In *Proc. 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, RAISE '18*, May 2018, 35–41. https://doi.org/10.1145/3194104.3194109

[18] M. Fowler. 1999. *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, Reading, MA.

[19] E. Gamma et al. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston.

[20] V. Gadiraju et al. 2023. "I wouldn't say offensive but…": Disability-Centered Perspectives on Large Language Models. FAccT'23: In *Proc. 2023 ACM Conf. on Fairness, Accountability, and Transparency*. June 2023, 205–216. https://doi.org/10.1145/3593013.3593989

[21] R. Gordon. 2023. Large language models are biased. Can logic help save them? MIT researchers trained logic-aware language models to reduce harmful stereotypes like gender and racial biases. MIT CSAIL, March 3, 2023. https://news.mit.edu/2023/large-language-models-are-biased-can-logic-help-save-them-0303

[22] Y.J. Ha et al. Exploring the impact of generative AI on the future of teaching and learning. https://cyber.harvard.edu/story/2023-06/impacts-generative-ai-teaching-learning

[23] S. Haijian et al. 2016. Algorithmic Bias: From Discrimination Discovery to Fairness-aware Data Mining. In P*roc. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16*. https://doi.org/10.1145/2939672.2945386

[24] C.E. Harris Jr. et al. 2019. *Engineering Ethics: Concepts and Cases*. Cengage Learning, Boston.

[25] O. Havazík and P. Pavlíčková, "How to design Agile game for education purposes in JIRA," 2020 IEEE/IFAC 7th International Conference on Control, Decision and Information Technologies (CoDIT), Prague, Czech Republic, 2020, pp. 331-334, doi: 10.1109/CoDIT49905.2020.9263937.

[26] Institute of Electrical and Electronics Engineers (IEEE). Software Engineering Body of Knowledge (SWEBOK). Retrieved. https://www.computer.org/education/bodies-of-knowledge/software-engineering.

[27] I. Jacobson et al. 1999. *The Unified Software Development Process*. Addison-Wesley, Reading, MA.

[28] N. Jastroch et al. 2011 Adapting Business and Technical Processes for Collaborative Software Development. In *Proc. 17th Int'l Conf. on Concurrent Enterprising (ICE 2011)*, 1-8. Aachen, Germany. INSPEC Accession Number: 12289921.

[29] Joint Task Force on Computing Curricula. Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. (23 February 2015) https://www.acm.org/binaries/content/assets/education/se2014.pdf.

[30] Joint Task Force on Computing Curricula. CS2023: ACM/IEEE-CS/AAAI Computer Science Curricula. CC2023: Software Engineering. https://csed.acm.org/wp-content/uploads/2023/09/SE-Version-Gamma.pdf.

[31] A. Kernbach et al. 2022. Design Thinking at a glance - An overview of models along with enablers and barriers of bringing it to the workplace and life. In *2022 26th International Conference Information Visualisation*, Vienna, Austria, 2022, 227-233. doi: 10.1109/IV56949.2022.00046

[32] V.D. Kirova, C.S, Ku, J.R. Laracy, T.J. Marlowe. Interactive Large Language Models and Software Engineering Workflows: Interactions and Implications. *In preparation.*

[33] C.K. Lo. 2023. What is the Impact of ChatGPT on Education? A Rapid Review of the Literature. *Education Sciences* 13, 4 (2023). https://doi.org/10.3390/educsci13040410

[34] R. Luckin and M. Cukurova. 2019. Designing Educational Technologies in the Age of AI: A Learning Sciences-Driven Approach. *British Journal of Educational Technology*, July 21, 2019. https://doi.org/10.1111/bjet.12861

[35] B.D. Mittelstadt et al. 2016. The Ethics of Algorithms: Mapping the Debate. *Big Data & Society* 3, 2 (December 1, 2016). https://doi.org/10.1177/2053951716679679

[36] R.S. Pressman. 2014. *Software Engineering: A Practitioner's Approach* (8th ed.) McGraw-Hill, New York.

[37] A. Radford et al. 2019. Language Models are Unsupervised Multitask Learners. OpenAI Blog. https://www.semanticscholar.org/paper/Language-Models-are-Unsupervised-Multitask-Learners-Radford-Wu/9405cc0d6169988371b2755e573cc28650d14dfe

[38] F.G. Rocha et al. 2021. Enhancing the Student Learning Experience by Adopting TDD and BDD in Course Projects. In *2021 IEEE Global Engineering Education Conference (EDUCON)*, Vienna, Austria, 2021, 1116-1125. doi: 10.1109/EDUCON46332.2021.9453916

[39] S.R. Schach. 2011. *Object-Oriented & Classical Software Engineering* (8th ed.) McGraw Hill, New York.

[40] K. Schwaber and M. Beedle. 2002. *Agile Software Development with Scrum*, Prentice Hall, Upper Saddle River, NJ.

[41] C.S. Smith. 2023. Hallucinations Could Blunt ChatGPT's Success. *IEEE Spectrum* (March 13, 2023). https://spectrum.ieee.org/ai-hallucination

[42] I. Sommerville. 2015. *Software Engineering* (10th ed). Pearson, London.

[43] A. Uche. 2023. 5 Reasons Why Companies Are Banning ChatGPT (June 26, 2023). https://www.makeuseof.com/reasons-why-companies-banning-chatgpt/

[44] E. Wickens and M. Janus. 2023. The Dark Side of Large Language Models, Part 2: "Who's a good chatbot?". *Hidden Layer* (March 24, 2023). https://hiddenlayer.com/research/the-dark-side-of-large-language-models-2/