



The Next Frontier in Software Development: AI-Augmented Software Development Processes

Ipek Ozkaya 

A fool with a tool is still a fool.

—Grady Booch

WITHIN A SHORT amount of six months, the productivity and creativity improvement promise delivered by generative artificial intelligence (AI), such as in large language models (LLMs), has taken the world by storm. This storm in particular is felt by developers and those conducting research in automated software engineering. The quick shift observed in software engineering conferences gives one data point to understand the magnitude of the excitement (and panic) the changes that the application of LLMs are hinting at. For example, the Automated Software

Engineering 2023¹ conference not only received a record number of 661 submissions, it received more than half of those focused on AI for software engineering topics, most of which experiment with various applications of LLMs in conducting software engineering tasks. The software engineering community has entered a “we don’t know what we do not know” period.

The challenge in front of every individual, team, and organization who is involved in the creation of software is an obligation to start figuring out what their expectations are from generative AI and shifting to AI-based tools. This cannot and should not be an exploration process driven by fear of missing out. What challenge will AI-based tools solve that today’s software development

tools and processes cannot address well? What will be the price of shifting to these generative AI tools, especially as they increasingly depend on foundation models in which a model trained on a large amount of unlabeled data can be adapted to many applications? What do we need to do to educate or hire the individuals with the right skill sets so that we can manage the risks involved and reap the ever so hoped for exponential benefits? We need to be reminded more strongly than ever before that despite the availability of potentially improved tools, it will be the humans who will use and guide these tools in their purposeful application. The infamous quote by Grady Booch, “a fool with a tool is still a fool,” is more relevant than ever as we enter this uncertain era of

Digital Object Identifier 10.1109/MS.2023.3278056
Date of current version: 14 July 2023

accelerated pace in development of AI-augmented software engineering tools and software development processes which rely on them.

The evolution of tools for software engineers to support improved efficiency in software development processes have historically focused on removing a significant barrier to enable better, faster, cheaper software development, resulting ideally in a higher quality product. Tools developed to support software engineers are powered by the vision to accelerate pace while reducing the number of mistakes. In this article, I will review how software development processes have evolved along with the tools that enable developers to execute them more effectively. I argue that, as we continue to investigate the potential of recent advances in LLMs and their applications in software development,² as well as other forms of automation with or without AI, such as bot-driven software engineering,³ we do not lose the focus of the key obstacles in quality and timely software system delivery that we are aiming to remove. As the software engineering community, we should be cautious to not create approaches whose creation and sustainment may create longer-term limitations. We should boldly and clearly recognize that removing challenges will not solely be achieved by the next most powerful generative AI model: it will require hybrid tooling fit for the task at hand.

AI-Augmented Software Development

AI-augmented software development refers to use of AI-based (while recognizing other tools are involved as well) automated tools to improve the efficiency of software engineers and reduce their cognitive load, shifting

the attention of humans to the conceptual tasks that computers are not good at and eliminating human error from tasks where computers can help. AI-augmented software development implies a multimodal human-computer “partnership” approach where the roles the software development tools and software engineers take can vary including but not limited to the following ways:

- an intern we don’t entirely trust but who does save us time, sometimes a lot of it
- a bot that does things for us³
- a partner or pair programmer that gives us advice.

With AI-based tools there are two paths that tools can take: do a task better without changing the flow of the task or complete the task with a different flow. AI-based tools provide opportunities in both aspects.

Improving developer productivity, consequently system quality, has been a key concern in software engineering for decades. A focus on improved automation, including AI-augmented tools (your favorite generative AI tool too), is neither new nor novel. Software development tools have already been incorporating use of AI-based tools for improved task execution. Examples are plentiful. Multiobjective search has demonstrated development of test cases, localization and triage of crashes, and suggesting and monitoring their fixes, such as Sapienz developed in Meta.⁴ Sapienz is used in production systems at Meta today. Natural language processing (NLP), such as bidirectional encoder representation from transformers (BERT), has been helpful in integrating information that is disconnected to improve requirement traceability.⁵ Often manual

CONTACT US

AUTHORS

For detailed information on submitting articles, visit the “Write for Us” section at www.computer.org/software

LETTERS TO THE EDITOR

Send letters to software@computer.org

ON THE WEB

www.computer.org/software

SUBSCRIBE

www.computer.org/subscribe

SUBSCRIPTION CHANGE OF ADDRESS

address.change@ieee.org
(please specify *IEEE Software*.)

MEMBERSHIP CHANGE OF ADDRESS

member.services@ieee.org

MISSING OR DAMAGED COPIES

contactcenter@ieee.org

REPRINT PERMISSION

IEEE utilizes Rightslink for permissions requests. For more information, visit www.ieee.org/publications/rights/rights-link.html

EDITORIAL STAFF

IEEE SOFTWARE STAFF

Journals Production Manager: Peter Stavenick,
p.stavenick@ieee.org

Cover Design: Andrew Baker

Peer Review Administrator: software@computer.org

Periodicals Portfolio Specialist: Cathy Martin

Periodicals Operations Project Specialist:

Christine Shaughnessy

Content Quality Assurance Manager: Jennifer Carruth

Periodicals Portfolio Senior Manager: Carrie Clark

Director of Periodicals and Special Products:

Robin Baldwin

IEEE Computer Society Executive Director:

Melissa Russell

Senior Advertising Coordinator: Debbie Sims

CS PUBLICATIONS BOARD

Greg Byrd (Interim VP of Publications), Terry Benzel,
Irena Bojanova, David Ebert, Dan Katz, Shixia Liu,
Dimitrios Serpanos, Jaideep Vaidya; Ex officio:
Robin Baldwin, Nita Patel, Melissa Russell

CS MAGAZINE OPERATIONS COMMITTEE

Irena Bojanova (Chair), Lorena Barba, Lizy K. John,
Fahim Kawsar, San Murugesan, Ipek Ozkaya,
George Pallis, Charalampos (Babis) Z. Patrikakis,
Sean Peisert, Balakrishnan (Prabha) Prabhakaran,
André Stork, Ramesh Subramanian, Jeff Voas

IEEE PUBLICATIONS OPERATIONS

Senior Director, Publishing Operations: Dawn M. Melley

Director, Editorial Services: Kevin Lisankie

Director, Production Services: Peter M. Tuohy

Associate Director, Information Conversion and

Editorial Support: Neelam Khinvasara

Senior Manager, Journals Production: Katie Sullivan

Senior Art Director: Janet Dudar

Editorial: All submissions are subject to editing for clarity, style, and space. Unless otherwise stated, bylined articles and departments, as well as product and service descriptions, reflect the author's or firm's opinion. Inclusion in *IEEE Software* does not necessarily constitute endorsement by IEEE or the IEEE Computer Society.

To Submit: Access the IEEE Computer Society's Web-based system, ScholarOne, at <http://mc.manuscriptcentral.com/sw-cs>. Be sure to select the right manuscript type when submitting. For complete submission information, please visit the Author Information menu item under "Write for Us" on our website: www.computer.org/software.

IEEE prohibits discrimination, harassment and bullying:

For more information, visit www.ieee.org/web/aboutus/whatis/policies/p9-26.html.

Digital Object Identifier 10.1109/MS.2023.3276830

and potentially slow process of code review has both benefitted from more targeted tool support, as well as investigations around automatic, flexible, and adaptive code analysis and recommendation of reviews using NLP and deep neural networks.⁶ Hierarchical clustering to learn past fix patterns is applied to drive program repair to automatically fix instances of common bugs.⁷ Last but not least, improved versions of LLMs have made auto code complete a reality despite pitfalls.⁷ All of these tools mimic the workflow of existing software engineering activities and aim to improve conducting them as both fast and correct.

The opportunity, and challenge, for the software engineering community is to discover whether the fast-paced improvements in AI assistants change how we engage with and orchestrate software development activities, such that the use of tools triggers changes in several dimensions. First and foremost, software engineers and business stakeholders do not just hope for modest improvement, but expect improvements at scale, reaching 10x or more reduction in resource need and error rates. They hope that such tools will reduce their reliance on highly skilled engineers, flipping the economies of scale in software development workforce challenges. Similarly, availability of better tools gives the illusion that the need for certain classes of activities, such as extensive testing and analysis, may become negligible. Such tools sadly can easily fool novice stakeholders that hard-to-find expertise will not be an as critical issue anymore.

There are several open questions: Will these new generation of AI-augmented tools be able to guarantee security, performance,

conformance to quality standards, and intended architectures? As tools get more sophisticated, will such tools help developers to manage ripple effects of decisions, manage complexity, and focus on difficult tasks? Will users (developers and end users alike) trust these tools? Will expertise demand decrease, as tools will carry some of the burden? Early empirical studies reflecting on use of LLM supported software development tools, such as Copilot, reflect that ability to critique will be a key skill needed, implying relying on expertise will not decrease.⁸ However, we will see the leap ahead to improvements desired and promised not simply by getting better tools that conduct current activity flows better, but also getting tools which help us redesign workflows, a step forward in evolving our software development processes.

A Brief, Incomplete History of the Evolution of Software Development Processes

Winston Royce is attributed with describing the implementation steps of software development process, infamously known as the *waterfall* process.⁹ Royce's depiction included the major activities of system and software requirements, analysis, design, coding, testing, and operations. Identifying and explaining a flow of software engineering activities enabled developers to get a handle on the otherwise complex to orchestrate software engineering projects. Computer-aided software engineering tools emerged to assist design modeling and implementation; requirement traceability tools got popularized to achieve consistency checking across artifacts



INTRODUCING THE “FAILURE MODE” COLUMN

With this July/August 2023 issue we are introducing a new column, “Failure Mode” by Lorin Hochstein, a senior software engineer at Netflix. This column is Lorin’s brainchild, taking root in his experiences in having had to deal with many normal and not so normal failures in the organizations where he worked. He is questioning what these failures can teach us about how we might succeed better.

If you have been in the business of software development, you know by now that failure in software is normal, unavoidable, and constant.

Despite this, software systems do work, some even work to the extent that they save our lives, safely take us to far places, enable us to communicate with one another over long distances, and more. Lorin will use failure as his subject matter to help us better understand software and how to write software that works better.

With Lorin assuming the “Failure Mode” column’s leadership, Silvia Abrahão and Miroslaw Staron will take on the helm of the “Practitioners’ Digest” column as its coeditors.



Silvia Abrahão is an associate professor at Universitat Politècnica de València, Valencia, Spain, and Miroslaw Staron is a professor in the Software Engineering Division, Chalmers University of Technology and the University of Gothenburg, Gothenburg, Sweden. Both Silvia and Miroslaw are not new to *IEEE Software*, having served as associate editors.

The goal of the “Practitioners’ Digest” column is to bring practice-relevant content from different venues and build bridges between communities. In their first column, along with their co-authors, they review research in Open Source Software: Communities and Quality, from a number of events in 2022.

I know our readers will enjoy both columns. As always, do reach out to us with your feedback, questions, and ideas.



to manage the ripple effects and dependencies between the “waterfall” software development activities. However, the unintended sequential execution of these activities created roadblocks due to delayed or nonexisting feedback loops and communication barriers.

The next generation of software development processes, with an effort to remove these barriers, focused on iterative and agile activity flows to address communication and feedback loop issues. Workflow management tools, along with integrated development environments and software

analysis tools, supported iterative and agile processes as the task orchestration changed from sequential on the entire requirements to iterative. Workflow management tools and issue trackers aimed to empower software engineers to work with a just-in-time mindset, pulling the next task based on their tempo to eliminate wait time and support iterative execution on requirements.

Software engineers learned the hard way that when not conducted with discipline, agile and iterative processes result in disconnects between design, development, and operations,

often causing unintended rework. No matter what process you are following, testing often gets short-changed. Variations of iterative and incremental processes that also incorporate “test first” philosophies further improved workflows, catching unintended mistakes early, supported by unit-testing frameworks.

DevOps philosophy and process further broke barriers between development and operations. DevOps deployment pipelines further enabled running automated testing, code review, and code analysis tools, whether they are AI-based or not.

This short summary of software development processes evolution is both incomplete and opportunistically focused on what worked. The key advances in software development processes and the automation that supports them have led us to better recognize and remove the barriers against improved productivity and more streamlined workflows. It is only when new workflows are also supported by purposeful automation that we start seeing significant improvements in productivity and quality.

stones to integrate continuously, as opposed to at the end of the development, and running all checks during each small integration, as opposed to during predetermined phases.

Tools enable removing barriers; however, the need for tools should be driven by where barriers exist and how workflows can be redesigned. Improved efficiencies are not solely a result of improved automation or process steps, but they are an outcome of designing workflows that target removing challenges.

code base and execute mundane but repetitive and expensive tasks. We need to ask what barriers we are removing with these tools and how can we redesign workflows to take advantage of AI-augmented and other tools for 10× improvements. What activities will be reordered? What activities will have less priority and what new activities and developer interaction models will be needed?

Once we better recognize what barrier we are removing, we can also maybe name this next exciting phase in the evolution of the software development processes. Maybe this next frontier is a “self-validating software development” process to emphasize that we may not need extensive assurance and validation, because we can catch and fix errors in real time and recommend how to write them correctly in the first place with our new tools. Perhaps we will want to call such a development process the “self-adaptive software development” process, as we will want to emphasize that our data are working for us and help reflect, improve time to conduct activities and their correctness, as well as where to focus. Or we will mirror this new frontier based on skills of expert engineers who know how to critique their own as well as other’s work, and will call it “reflective, intelligent software development” or “design prompt-driven iterative development.”

In this brave new world, we should be reminded early on and often enough that AI is the means to the end, not the goal. It is a tool, like many others that we have been benefiting from. We will accomplish improvements only when we map how the tools remove the barriers, not when we develop better foundation models, general AI, or the next

In this brave new world, we should be reminded early on and often enough that AI is the means to the end, not the goal.


The shift from waterfall to agile, iterative, and incremental processes was enabled by recognizing that if software engineers conduct all tasks around a smaller scope of requirements, and then iterate and incrementally grow, as opposed to tackling the entire scope, conducting all activities consecutively, they can identify and resolve problems earlier. A new generation of tools supported this new smaller scope of task orchestration.

Writing tests first, implementing against them, and running all tests as the system is developed, as opposed to writing and running tests at the end of the requirement, analysis, design, and implementation steps, removed barriers in wait time until errors were discovered. Having tests written first in automated frameworks served as one of the stepping

The Next Frontier

We are at a turning point where AI-based approaches to software development automation will empower improvements on several fronts, including correctness, scale, and timeliness, despite many potential risks and pitfalls.² The risks will likely initially drive up the costs of development and sustainment. However, 10× improvement will not be an outcome of better tools, even if they help; 10× improvement will be an outcome of understanding and redesigning task flows to remove barriers.

The question in front of the software engineering community is not how to develop the best LLM to become the next autocoder, or the search-based reasoning tool to enable tradeoff analysis at the pareto front, or the bot that can crawl a

best chat bot, which answers any programming questions. 

References

1. “ASE 2023,” in *Proc. 38th IEEE/ACM Int. Conf. Automated Softw. Eng.*, Kirchberg, Luxembourg, Sep. 2023. [Online]. Available: <https://conf.researchr.org/home/ase-2023>
2. I. Ozkaya, “Application of large language models to software engineering tasks: Opportunities, risks, and implications,” *IEEE Softw.*, vol. 40, no. 3, pp. 4–8, May/Jun. 2023, doi: 10.1109/MS.2023.3248401.
3. I. Ozkaya, “A paradigm shift in automating software engineering tasks: Bots,” *IEEE Softw.*, vol. 39, no. 5, pp. 4–8, Sep./Oct. 2022, doi: 10.1109/MS.2022.3167801.
4. N. Alshahwan et al., “Deploying search based software engineering with Sapienz at Facebook,” in *Proc. Int. Symp. Search Based Softw. Eng. (SSBSE)*, 2018, pp. 3–45.
5. J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, “Traceability transformed: Generating more accurate links with pre-trained BERT models,” in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, 2021, pp. 324–335, doi: 10.1109/ICSE43902.2021.00040.
6. A. Gupta and N. Sundaresan, “Intelligent code reviews using deep learning,” presented at the Deep Learning Day, London, U.K., Aug. 2018.
7. J. Bader, A. Scott, M. Pradel, and S. Chandra, “Getafix: Learning to fix bugs automatically,” *Proc. ACM Program. Lang.*, vol. 3, no. OOPSLA, Oct. 2019, Art. no. 159, doi: 10.1145/3360585.
8. C. Bird et al., “Taking flight with copilot: Early insights and opportunities of AI-powered pair-programming tools,” *Queue*, vol. 20, no. 6, pp. 35–57, Nov./Dec. 2023, doi: 10.1145/3582083.
9. W. W. Royce, “Managing the development of large software systems,” *Proc. IEEE WESCON*, vol. 26, pp. 1–9, Aug. 1970.



IEEE COMPUTER SOCIETY
Call for Papers

Write for the IEEE Computer Society's authoritative computing publications and conferences.

GET PUBLISHED
www.computer.org/cfp

 IEEE COMPUTER SOCIETY

 IEEE