Editor in Chief: **Ipek Ozkaya**
Carnegie Mellon Software Engineering Institute
ipek.ozkaya@computer.org

# A Paradigm Shift in Automating Software Engineering Tasks: Bots

Ipek Ozkaya

**INCREASING DEVELOPER PRODUCTIVITY** and improving software quality are enduring challenges in software engineering. Addressing these challenges has influenced research toward improved automation and formalisms to realize software development tasks, along the way seeking for disruptive ways to support software developers. The past 55 years of software engineering has seen a number of paradigm shifts, fundamental changes in our underlying assumptions which led to making some progress in developing better quality software and disruptive improvements. For example, agile and lean software development approaches challenged our assumptions about the way software development tasks are orchestrated and communicated. Agile and lean software development philosophies advocate elimination of waste by emphasizing value-added tasks and encouraging improved understanding of what tasks stay in inventory.[1] Embracing agile and lean software development approaches with discipline led software engineers to prioritize software engineering activities with a bias towards visible delivery, most often leading to improved software quality and developer well being when done right.

In an effort to disrupt the challenge of improved software quality and developer productivity there has also been a focus on increasing amount of automated support throughout the software development lifecycle. Many of the automation and tool support developed during the previous decade for model-based software engineering,[2] continuous integration and deployment,[3] defect and vulnerability analysis,[4] automated bug fixing,[5] modern code review,[6] and value stream management[7] were all developed with the goal of improving software development efficiency and quality. The tooling philosophy used in all of the aforementioned techniques has been to solve a complex, large software development task with a dedicated application, which then is added onto the software development toolchain. In all these tools, software engineers are in the driver's seat while their common goal is to shift the attention of developers to the conceptual tasks that computers are not good at and eliminate developer error from tasks where computers can help.

Providing increasingly capable automated environments to software engineers has been and will continue to be a top priority in software engineering. Correctness, scale, trust, and user overhead are often among the top concerns that need to be addressed. Although improved tools have been developed, ranging from static code analyzers with fewer false positives, to model-based environments with improved code generation capabilities, to auto-generated tests to give a few examples, progress has still been limited. Automation efforts are often targeted at improving what already exists, rather than rethinking how automated development tasks should work, maybe even with a different workflow. Automated

*Digital Object Identifier 10.1109/MS.2022.3167801*
*Date of current version: 22 August 2022*

software engineering needs a disruptive perspective, a paradigm shift, where we challenge the underlying assumptions of what needs to be and can be effectively automated in software development.

Software development bots, along with increasing applicability of artificial intelligence (AI) and machine learning (ML) techniques, do offer an opportunity for researchers and tool vendors to revisit the fundamental assumption of what can be automated in software engineering and how. There is applicability of software engineering bots at almost every stage of software development, holding the promise of assisting software engineers and making the process more efficient, effective, and enjoyable. Envisioning software development tool chain with assistance from bots will find broader acceptance by developers and reach wider adoption if they are scoped correctly. Next generation automated software engineering, including bots, should narrowly focus on workflows that target data-intensive and tedious activities, which might potentially shift handoffs and task dependencies in the software development lifecycle. Researchers and tool developers need to shift their perspective to make speedy progress, initially driven by the following goals:

- *Think small*: Solve-bounded tasks, but at scale with improved speed and correctness, rather than solve complex tasks with several dependencies.
- *Eliminate tasks*: Identify and automate the tasks that can be mostly removed from the developer's responsibilities, rather than add new tasks even if automated or simply replace them with automation but still requiring developers to closely monitor.
- *Empower partnership*: Enable a developer-tool-teaming model, rather than developer-tool-interaction model.

An underlying message in articulating these goals is to also remind researchers and tool vendors to talk to developers more often to better understand challenges and adoption models. Empirical studies with developers have identified that there is a tendency to perceive any application as a bot and include a large range of characteristics.[8,9] There is room for a range of approaches and objectives for improving automated tool support in software engineering; however, development bots when scoped around well defined goals fill a unique gap and can create leap ahead improvements.

## Think Small

Software development bots, commonly defined as *automated tools that handle tedious tasks*, force their creators to scope automation goals more narrowly. What constitutes a tedious task can have a wide range of characteristics. Tedious tasks are repetitive, numerous, and have bounded decision space. These criteria, when applied to automation, narrow application scenarios of automated tools while potentially increasing their likelihood of providing correct outcomes. For example, a static analysis tool checks for a range of software quality conformance criteria, but not all conformance rules have applicability to the entire codebase or have well-defined criteria. Creating bots for those repetitive, numerous, and well-defined conformance violations and integrating them into the development flow as default checkers

**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

improves achieving just-in time code corretness. The ability to provide tool support becomes a more tractable problem as responsibilities shift, where tedious checks are provided as bots and the remaining tasks can be addressed with different techniques, including automation but not limited to.

The repetitive, numerous, and bounded decision-space criteria provide opportunities for AI- and ML-based approaches to be more successful. Repetitive and numerous tasks also tend to have more data, which can enable development of AI assistants.

Most developers do not enjoy repetitive tasks at scale due to their lack of creativity in task execution. However, decomposing complex tasks into smaller ones that are tedious in nature can provide more opportunities of developing impactful automation. For example, software evolution and refactoring are complex tasks with a lot of decisions involved, however, enumerating dependencies and categorizing them are not—these tasks are also common to many other software development activities.[10] In this new paradigm, automation becomes a quest for rethinking software development as a series of tedious and common tasks and automating them, rather than mimicking the complicated developer tasks with automation.

## Eliminate Tasks

Incorporating bots and AI-based developer-support tools need to also include envisioning more effective workflows for developers, where tedious tasks are offloaded from the developer's responsibilities. For example, improved real-time assistance in code-quality conformance can reduce reliance on the added static code analysis checks during testing and deployment, improving the effort balance of local conformance analysis versus analysis of system-wide, cross-cutting, and harder-to-conduct architectural quality and runtime concerns. Today, system-wide analysis is often an activity which requires time consuming manual intervention despite tool support. With improved testing bots, software engineers may not need to test for certain classes of bugs when AI-augmented bug fixing becomes a trusted service at scale.

Can automation be scoped to shift software development tasks from developers to tools? Can developers and users trust systems supported by such automation? Re-envisioning automation as small chunks of tedious tasks can enable improving trust and developer task overhead as well. Narrowly scoped tasks that follow repetitive, numerous, and well-defined decision-space criteria are often easier to validate due to ability to sketch decision options supported by data. Trust will be built as automation not only supports developers but surpasses their ability to achieve the capabilities of being able to resolve tasks better and faster than humans, and without errors due to cognitive decline. As the capabilities of these narrowly scoped services increase, they also will shift responsibility from developers to the tools. If each bot, automated service, is added as yet another new tool to the development toolchains, the places where developers need to integrate their development data will increase, complicating efforts, only with automation. Achieving success in a

*Digital Object Identifier 10.1109/MS.2022.3154846*

## A “REQUIREMENTS” COLUMN UPDATE

Change is the only constant! At *IEEE Software*, we try hard to embrace change and thrive to make it part of our success, even if at times it is hard to accept. Sarah Gregory, who since 2017 has successfully led one of our flagship columns, “Requirements,” has decided to step down. Sarah, as a requirements and systems engineer at Intel Corporation, has shared her insights and experiences through her 21 articles published in *IEEE Software*. She helped connect research in requirements engineering to practice based on her experience as well as inviting others to share their experiences. We are grateful for her service and thank her for her contributions.

Starting with the November/December 2022 issue, Markus Borg, a principal researcher at CodeScene, will be taking the helm of the “Requirements” column. Previously, Markus was a senior researcher with RISE Research Institutes of Sweden. He is also an adjunct lecturer at Lund University from where he obtained a Ph.D. in 2015. His research interests include requirements engineering, software testing, verification and validation, and safety engineering. Currently, Markus’ research primarily targets quality assurance of applied artificial intelligence in the automotive domain. Before embarking on the career in research, Markus worked at ABB as a software engineer in process automation. Markus serves on the editorial board of *Empirical Software Engineering* and is a board member of Swedsoft, an independent nonprofit organization with the mission to increase the competitiveness of Swedish software. He is an IEEE Member. He is especially looking forward to bringing his testing and AI perspectives into how to improve requirements engineering practices. Welcome aboard, Markus!

new automated software engineering paradigm will necessitate seamless integration with development environments and empowering developer-tool partnership.

## Empower Partnership

When narrowly scoped tasks are offloaded to tools as services, mostly AI-based bots, both the developers and these AI-based bots as assistants will need to have a supervisory role. Developers need to guide and consequently improve bots’ capabilities. And AI-based bots eventually will take on a supervisory role by providing real-time feedback and, in time, demonstrating repeated mistakes to developers. There will always be some developers on a team whom you trust more than others, perhaps due to experience, skill sets, or demonstrated performance. Bots as AI-assisted development workflows will trigger the need to think these tools as “partners” in the same way. The partnership developers and automated assistants will engage in as part of an overall team that produces software of sufficient quality will range from complete delegation to software development bots to bots providing feedback and improving based on developer input.

These interactions need to be envisioned as developer-tool teaming rather than developer-tool interaction, although there are relevant scenarios for both. The trust in such automated solutions, whatever their form, boils down to a risk assessment. What’s the probability that the result is correct? What is the overhead and complexity involved in detecting a wrong decision? What’s the impact if it’s wrong, and how can the associated risks be dramatically reduced if the bot makes the wrong decision? Developers will initially have to take the driver’s seat and initial progress will be slow, however, advances in answering such questions will accelerate progress.

We will likely see a head-spinning pace of bots and AI-augmented tools in the next decade to support software development, driven not only by research but also by their rapid integration into existing developer tools. For example, the release of GitHub Copilot already triggered studies which concluded that, although more lines of code may be produced by such AI-assisted automation, potentially implying increased productivity, the quality may not necessarily be always as expected.[11] This balance will improve in time.

The assistance provided by automated tools, including in the form of bots, is not new in software engineering. The goal of creating a collaboration model that improves transparency of the development workflow through connecting tools, processes, and automation, is also now clearly associated with bots.[12] Some readers may rightly ask, "Where is the paradigm shift?" The paradigm shift is the opportunity that bot- and AI-augmented automation provides in challenging how we scope automation. We need to embrace the tediousness of software development as an opportunity. We need to seek all those tedious tasks hidden in the complex design, search, tradeoff analysis, verification and validation, and many other decision-making activities of developers. In doing so we open the door to a new reality in automated software development where trust is easier to build because tasks are easier to validate and developers have different tasks because the mundane ones are offloaded to the tools. Regardless, developers have the control in the partnership to guide improvement of the tools.

Repeat after me! Software development is composed of many tedious activities with countless opportunities for trustworthy and reliable automation.

## References

1. D. Reinertson, *The Principles of Product Development Flow: Second Generation Lean Product Development.* Celeritas Publishing, 2019.
2. P. Lago, I. Malavolta, H. Muccini, P. Pelliccione, and A. Tang, "The road ahead for architectural languages," *IEEE Softw.*, vol. 32, no. 1, pp. 98–105, Jan./Feb. 2015. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6756703, doi: 10.1109/MS.2014.28.
3. A. Rahman, A. Partho, D. Meder and L. Williams, "Which factors influence practitioners' usage of build automation tools?" in *Proc. IEEE/ACM 3rd Int. Workshop Rapid Continuous Softw. Eng. (RCoSE)*, 2017, pp. 20–26, doi: 10.1109/RCoSE.2017.8.
4. P. Morrison, R. Pandita, Z. Xiao, R. Chillarege, and L. A. Williams, "Are vulnerabilities discovered and resolved like other defects?" *Empirical Softw. Eng.*, vol. 23, no. 3, pp. 1383–1421, Jun. 2018, doi: 10.1007/s10664-017-9541-1.
5. C. Le Goues, M. Pradel, and A. Roychoudhury, "Automated program repair," *Commun. ACM*, vol. 62, no. 12, pp. 56–65, 2019. [Online]. Available: https://cacm.acm.org/magazines/2019/12/241055-automated-program-repair/fulltext, doi: 10.1145/3318162.
6. C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: A case study at google," in *Proc. 2018 IEEE/ACM 40th Int. Conf. Softw. Eng., Softw. Eng. Practice Track (ICSE-SEIP)*, pp. 181–190, doi: 10.1145/3183519.3183525.
7. G. C. Murphy, M. Kersten, R. Elves, and N. Bryan, "Enabling productive software development by improving information flow," in *Rethinking Productivity in Software Engineering*, S. Caitlin and Z. Thomas, Eds. Berkeley, CA, USA: Apress, 2019, pp. 281–292.
8. L. Erlenhov, F. Gomes de Oliveira Neto, and P. Leitner, "An empirical study of bots in software development: Characteristics and challenges from a practitioner's perspective," in *Proc. 28th ACM Joint Meeting on Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2020, pp. 445–455, doi: 10.1145/3368089.3409680.
9. C. Lebeuf, A. Zagalsky, M. Foucault, and M. Storey, "Defining and classifying software bots: A faceted taxonomy," in *Proc. 1st Int. Workshop Bots Softw. Eng.*, 2019, pp. 1–6, doi: 10.1109/BotSE.2019.00008.
10. J. Ivers, C. Seifried, and I. Ozkaya, "Untangling the knot: Enabling architecture evolution with search-based refactoring," in *Proc. 19th Int. Conf. Softw. Architecture*, 2022, pp. 101–111, doi: 10.1109/ICSA53651.2022.00018.
11. S. Imai, "Is GitHub copilot a substitute for human pair-programming? An empirical study," in *Proc. 2022 IEEE/ACM 44th Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, pp. 319–321, doi: 10.1109/ICSE-Companion55297.2022.9793778.
12. M. A. Storey and A. Zagalsky, "Disrupting developer productivity one bot at a time," in *Proc. 2016 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, pp. 928–931, doi: 10.1145/2950290.2983989.