



Are DevOps and Automation Our Next Silver Bullet?

Ipek Ozkaya

FRED BROOKS, IN his well-known classic *The Mythical Man-Month*, already told the software engineering industry in 1975 that there are no silver bullets in gaining an order-of-magnitude improvement in software productivity.¹ He also observed that “most of the big past gains in software productivity have come from removing artificial barriers that have made the accidental tasks inordinately hard, such as severe hardware constraints, awkward programming languages, lack of machine time.” The hope and goal of software development processes in orchestrating the essential and accidental software engineering and development tasks is precisely to remove artificial barriers to delivering better, faster, cheaper software to the users. Our next silver bullet seems to have emerged as automating repeatable, manual process

tasks. While, on one hand, we debate how to scale agile, on the other, we run to DevOps, continuous integration, and continuous delivery tools to achieve the so-called orders of magnitude of productivity improvement.

Automation enshrines a process step without necessitating further communication if the process step is well understood and the automated solution addresses its goal on target. The place for automation is well-bounded, repetitive tasks. In fact, the DevOps movement lives on the mantra, “If you do a task more than twice, automate it.” Automation facilitates concrete communication through artifacts, saving time in handovers as well as misinterpretations.

Bass et al.² define DevOps as “a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality.” DevOps is not a

software development process, but it supplements these processes to eliminate time wasted on handovers between tasks, eliminating the need for accidental activities. DevOps delivers value of collaboration between development and operations staff throughout all stages of the development lifecycle and supports this through tools and infrastructure. The ability to automate aspects of this collaboration not only saves on eliminating inconsistencies, it also promises to achieve the ever-so-longed-for speed of development and propagating change—the silver bullet.

Over the past several decades, we have seen many “next best” processes rise and decline in popularity: waterfall, Rational Unified Process, Extreme Programming, Scrum, Lean, Kanban, test-driven development, and all of the various frameworks to scale Scrum in larger contexts, such as Scaled Agile Framework,³ Nexus,⁴ and Large-Scale Scrum.⁵ With the

Digital Object Identifier 10.1109/MS.2019.2910943
Date of publication: 18 June 2019

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

help@computer.org

REPRINT PERMISSION

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

abundance of methodologies to choose from, process improvement initiatives, such as Capability Maturity Model Integration in its heyday, helped hundreds of organizations assess and evolve their software development processes. Software development organizations learned some valuable lessons along the way:

- The higher the level of ceremony involved in the software development process, it becomes increasingly likely that the process will not achieve its intended productivity consequences.
- No process definition will magically deliver results without a committed, disciplined team, and organization behind it.
- Context is key; there is no “one size fits all” solution.

Almost two decades after “Agile Manifesto”⁶ was written, we are still debating what it really means to be agile, how to scale agile software development processes and combine them with other development processes, and whether hybrid processes are the way to go. I will argue here, reflecting Brooks’s observation, that any of the successes we have seen in the software development processes, in particular, agile and lean software development processes, were precisely due to their success in removing barriers, such as the overhead from document-driven approaches being replaced by working software demonstrations, which helped to eliminate some accidental tasks. Many agile software development success stories have been attributed to the adoption of practices such as increased team communication, collective ownership, frequent customer-visible releases, backlog-driven requirements

management, continuous integration, and shorter iterations.⁷

Yet still, failures are abundant, even with all the guidance and lessons learned from case studies. The journey to defining and following a golden software development process still has not delivered on the promise of order-of-magnitude improvement, leaving software organizations in the fog of war with two enduring challenges: *customization to context* and *communication*. The solution to these challenges is not in defining the next software development process or scaling our favorite one up. In fact, any of the existing iterative, incremental software development processes will do just fine as long as we understand the context we are developing in and the essential communication channels that need to be managed. DevOps and automation can help further by removing accidental mistakes through incorporating tools of our trade to leave precious time for the most essential manual-effort-requiring tasks, such as design.¹

Customization to Context

At a minimum, the business environment, team, governance structure, criticality of the system, size and age of the system, and rate of change will influence how to orchestrate any software development process for its context. Software development processes, out of their box, do not always fit their context, while most of them use the same building blocks. The software industry learned this lesson during its journey through embracing agile software development. The software factories paradigm of the 1980s failed to deliver its promises. Many scaled agile-development endeavors failed due to assuming that scaling simply meant a matter of tailoring Scrum to work with larger

groups of people.⁸ The poster-child examples of such failures were often in the context of dependable software systems where high assurance and compliance requirements dominated, such as avionics, financial industry, or embedded command-and-control software. Ad hoc cherry picking of activities, for example, agile says design emerges, so we do not spend time architecting, were at the root of many failures.

The lesson, going forward, is no matter what the choice of process is, you will need to understand and map the essential characteristics of your selection to the essential characteristics of your context. It is often not the process that does not scale or fails; more often than not, it is its interpretation and application.

Communication


A common goal of all processes is to eliminate nonessential activities that consume resources away from development and delivery of functionality that provides quality to its users. Software development processes define roles, responsibilities, and activities, with the ultimate goal of orchestrating these activities efficiently, mapping them to people and time. The activities of understanding business goals; grasping and specifying requirements; comprehending the data, modeling, designing, developing, and testing; and deploying functionality to the targeted users do not change. How these essential activities are ordered and how information flows between roles are where the processes differ. Software development processes that replace communication with over-the-wall tasks consequently fail. Software development processes that encourage frequent and on-task communication among all stakeholders and team members

receive increased adoption. Agile and lean software development processes exemplify this.

Again, the software organizations learned their lesson the hard way here. A process definition cannot eliminate misunderstood requirements, customers who keep changing their minds, and team members who do not agree on technical tasks.

The Role of DevOps and Automation

In our quest for the silver bullet since 1975, software engineering organizations definitely have achieved some successes, in particular, in transforming to agile development processes.⁸ Organizations that eliminate waste and inconsistencies through targeted automation of handovers and activities do see the benefits of reduced cycle time and resource expenditure. Whatever new process definition we may see in the future is likely to build on reaping the benefits of continuing to automate more of the lifecycle activities.

But has the automation promised through DevOps helped to achieve the order of magnitude of productivity improvement? Some accounts coming out of organizations adopting effective tooling and continuous delivery infrastructures describe gains in reducing rework costs,¹⁰ while others also report on overheads and failures as well. Failure stories are often similar: not understanding the context, dropping key activities between communication barriers, failing to collaborate—none of these activities can be automated, and that is where all processes fail in their adoption. As Brooks told us in 1975, there is no silver bullet. Neither DevOps nor automation is one, either. 

(continued on page 95)

EDITORIAL STAFF

IEEE SOFTWARE STAFF

Managing Editor: Jessica Welsh, j.welsh@ieee.org

Cover Design: Andrew Baker

Peer Review Administrator: software@computer.org

Publications Portfolio Manager: Carrie Clark

Publisher: Robin Baldwin

Senior Advertising Coordinator: Debbie Sims

IEEE Computer Society Executive Director: Melissa Russell

CS PUBLICATIONS BOARD

Fabrizio Lombardi (VP for Publications), Alfredo Benso, Cristiana Bolchini, Javier Bruguera, Carl K. Chang, Fred Douglass, Sumi Helal, Shi-Min Hu, Sy-Yen Kuo, Avi Mendelson, Stefano Zanero, Daniel Zeng

CS MAGAZINE OPERATIONS COMMITTEE

Sumi Helal (Chair), Irena Bojanova, Jim X. Chen, Shu-Ching Chen, Gerardo Con Diaz, David Alan Grier, Lizy K. John, Marc Langheinrich, Torsten Möller, David Nicol, Ipek Ozkaya, George Pallis, VS Subrahmanian

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 Managing Editor: Geraldine Krolin-Taylor

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. Articles must be original and not exceed 4,700 words including figures and tables, which count for 200 words each.

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.2019.2910944

- Boeing 737MAX,” Mar. 18, 2019. [Online]. Available: <http://newsinflight.com/2019/03/18/the-updating-changes-to-mcas-software-on-boeing-737max/>
11. S. Masson, “Volkswagen admits 17,000 complaints about dieselgate fix fails,” Dec. 7, 2017. [Online]. Available: <https://www.thecarexpert.co.uk/volkswagen-17000-complaints-dieselgate/>
 12. P. Mellor, “CAD: Computer-aided disaster,” *High Integrity Syst.*, vol. 1, no. 2, pp. 101–156, 1994.
 13. A.-F. Rutkowski and C. Saunders, *Emotional and Cognitive Overload: The Dark Side of Information Technology*. Evanston, IL: Routledge, 2019.
 14. D. Yadron and D. Tynan, “Tesla driver dies in first fatal crash while using autopilot mode,” June 30, 2016. [Online]. Available: <https://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk>
 15. T. Mann and S. Hughes, “Fast-tracked aircraft certification, pushed by Boeing, comes under the spotlight,” *Wall Street J.*, Mar. 24, 2019. [Online]. Available: <https://www.wsj.com/articles/fast-tracked-aircraft-certification-pushed-by-boeing-comes-under-the-spotlight-11553428800>
 16. A. Pasztor, A. Tangel, R. Wall, and A. Sider, “How Boeing’s 737 MAX failed. The plane’s safety systems, and how they were developed, are at the center of the aerospace giant’s unfolding crisis,” *Wall Street J.*, Mar. 27, 2019.
 17. J. A. Whittaker, *How to Break Software*. Reading, MA: Addison-Wesley, 2002.
 18. N. Jones et al., “Conversion from robotic surgery to laparotomy: A case-control study evaluating risk factors for conversion,” *Gynecol. Oncol.*, vol. 134, no. 2, pp. 238–242, 2014. doi: 10.1016/j.ygyno.2014.06.008.
 19. G. Travis, “How the Boeing 737 Max disaster looks to a software developer,” *IEEE Spectr.*, Apr. 10, 2019. [Online]. Available: <https://spectrum.ieee.org/aerospace/aviation/how-the-boeing-737-max-disaster-looks-to-a-software-developer>

FROM THE EDITOR

(continued from page 5)

References

1. F. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, 2nd ed. Reading, MA: Addison-Wesley, 1995.
2. L. J. Bass, I. M. Weber, and L. Zhu, *DevOps: A Software Architect’s Perspective*. Reading, MA: Addison-Wesley, 2015.
3. SAFe. Accessed on: May 7, 2019. [Online]. Available: <https://www.scaledagileframework.com>
4. Scrum.org, “An introduction to the Nexus framework.” Accessed on: May 7, 2019. [Online]. Available: <https://www.scrum.org/resources/introduction-nexus-framework>
5. LeSS. Accessed on May 7, 2019. [Online]. Available: <https://less.works>
6. Agile Manifesto. Accessed on: May 7, 2019. [Online]. Available: <https://agilemanifesto.org>
7. S. Bellomo, R. L. Nord, and I. Ozkaya, “A study of enabling factors for rapid fielding: Combined practices to balance speed and stability,” in *Proc. 35th Int. Conf. Software Engineering (ICSE 2013)*, San Francisco, CA, pp. 982–991.
8. W. Hayes, M. A. Lapham, S. Miller, E. Wrubel, and P. Capell, “Scaling agile methods for Department of Defense programs,” Carnegie Mellon Univ., Pittsburgh, PA, Rep. CMU/SEI-2016-TN-005, 2016. [Online]. Available: https://resources.sei.cmu.edu/asset_files/TechnicalNote/2016_004_001_484647.pdf
9. M. Paasivaara, B. Behm, C. Lassenius, and M. Hallikainen, “Large-scale agile transformation at Ericsson: A case study,” *Empirical Software Eng.*, vol. 23, no. 5, pp. 2550–2596, 2018. doi: 10.1007/s10664-017-9555-8.
10. Google Cloud, “New research from DORA: What sets top-performing DevOps teams apart.” Accessed on: May 7, 2019. [Online]. Available: <https://cloudplatformonline.com/2018-state-of-devops.html>



IEEE COMPUTER SOCIETY

DIGITAL LIBRARY

Access all your IEEE Computer Society subscriptions at computer.org/mysubscriptions