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

Crossing the Great Divide of Software Engineering

Ipek Ozkaya

THE GREAT DIVIDE, the continental divide of the Americas, is a water divide extending from the Bering Sea to the southern tip of South America, the Strait of Magellan. The Great Divide separates the watersheds that drain into the Pacific Ocean from those river systems that drain into the Atlantic and Arctic Oceans. While there are many other examples of geological and geographical divides, the Great Divide stands out by its magnitude and higher peaks compared to other divides and its spanning multiple countries and continents. The divide does not necessarily define any different characteristics on either of its sides; it is simply a water passage dividing the Americas into two. To hiking enthusiasts, hiking the Great Divide is defined by its magnitude, typically easily taking up to five months. Due to its clear visualization power, the Great Divide has been a source of inspiration for metaphors in movies, albums, books, and topics when describing any conundrum with two sides.

Software engineering has a fair share of its divides as well. Most of these divides close in time as everyone's level of knowledge increases, the conflicting technical aspects get resolved, all involved get trained in new techniques, and criteria for when to apply them are clarified. The divides of agile and architecture;[1] waterfall and iterative processes;[2] or formal (for example, using rigorous model semantics and expression) and semiformal (for example, unified modeling language) design techniques[3] are some examples. Yet none of these divides have been as monumental and as difficult to navigate as the one between the "researcher" and "practitioner" separation that the software engineering community has self-imposed.

In software engineering conferences and software research organizations a top priority concern is how research can impact and improve the practice of software engineering. When we look at organizations that hire software engineers in masses, their top concern is to hire those who have the expert knowledge of the most relevant and current software engineering techniques to solve the organization's most challenging problems. So the goals on either side of the divide are not that misaligned—to develop software and develop it well—yet ongoing conversations revolve around the disconnect, in particular the lack of practical and timely relevance of software engineering research.[4]

In this editorial, I invite you all, regardless of if you identify as a researcher or practitioner, to gear up and walk this Great Divide of software engineering with me. My goal is to convince you all that one step toward crossing this divide effectively and making it less daunting is to be all reminded that first and foremost, we all are *software engineers* regardless of our current roles and responsibilities. We need to collectively shift our attention to contributing to a common goal for the profession of software engineering rather

Digital Object Identifier 10.1109/MS.2022.3161770
Date of current version: 20 June 2022

than focusing on the researcher versus practitioner divide.

Common Goals for the Profession

Our experience growing up affects our thinking patterns. I grew up in a family of medical doctors. Some had private practices, some worked in device or pharmaceutical companies, and some were employed in research universities raising the next generation of doctors. Those who had a private practice or were employed at device or pharmaceutical companies definitely had different activities and financial incentives compared to those who were employed in research universities. Yet no matter how different their days and activities looked, they all converged in one common goal: how the outcomes of all their day-to-day activities contributed to improving the health of individuals and saving lives.

Given how ubiquitous software has become, a similar analogy to the medical practice can easily be made for software engineering. No matter what the nature of the activities we all engage in, the common goal is, or should be, to develop software that is safe, secure, and meets its resource and business goals for the domain it serves. Activities engaged by software engineering research groups and those executed by organizations in principle have this common goal.

The ability to make progress toward a common goal for the profession, which I defined as developing safe and secure software that meets its resource and business goals for the domain it serves, requires understanding the success criteria for the goal. For example, in a survey Lo et al. conducted with 3,000 software developers, they found no correlation between citation counts of research articles (an essential metric used in promotion cases) and the surveyed developers suggesting that the work is relevant to software engineering practice.[5] Koziolek, in a recent article, similarly emphasized that the practical impact of research requires different success measurement criteria.[6]

Working toward a common profession goal requires thinking like a software engineer before a practitioner or researcher. As a software engineer, success is not how many other people have read a paper, but it is how many other people used a developed technique to develop new functionality better. As a software engineer, success is not how quickly I shipped functionality that works, but it is how I shipped functionality using techniques that help guarantee that it not only works but also does not compromise any safety, security, or privacy issues. Aligning the success criteria around a commonly agreed-upon set of "goodness" principles of software is an important place to start.[7]

Building Skills to Cross the Divide

Not all software engineers have to have the skills to conduct long-term fundamental research or be cognizant of all the development frameworks and tools to ship software. There is simply too much to know; technologies and tools in software engineering change quickly; and theories start breaking as new techniques, tools, and hardware are introduced. Specialization matters, and an expert in secure coding will not necessarily be as well versed in optimizing performance. However, all software engineers need to know where to start. All software engineers need to know how to build a hypothesis relevant to software engineering

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

Digital Object Identifier 10.1109/MS.2022.3148369

challenges and collect evidence to prove it one way or another either with a prototype or by collecting and analyzing data. All software engineers need to know the software engineering lifecycle and key activities and the body of knowledge.[8] Building such a fundamental basic set of skills sets the stage for not only effectively crossing the divide but also understanding what it takes to survive on the other side.

However, not enough software engineers cross the divide between research and practice, starting their careers on one side and continue on the other side. The ability to build a career that crosses the divide requires having the skills that are fundamental to the tasks of a researcher and a practicing engineer as well as having bridges that make this transition easy. The career paths in software engineering are not welcoming in making individuals consider choices that involve both research and practice. This further contributes to the divide. Researchers become disconnected from practice with only options left to further focus on acquiring minor delta research funding and putting out more publications. This is also further encouraged by the structure of academic promotions. Practitioners as a result of the ever-increasing resource challenges need to focus on delivering on tasks and become disconnected from scoping enduring challenges that may require several hypothesis and test cycles. Of course, some of the save the day engineering strategies do end up solving difficult problems, and some research do make it into practice and change the way how developers work. However, regardless of success, the examples where the work is conducted collaboratively is simply not frequent enough.

Be a Software Engineer Before a Researcher or a Practitioner

I argue that our ability to bring software engineering research and practice closer will not be possible if we continue to embrace our career roles (researcher or practitioner) before we embrace our profession role (software engineer). We all need to think and act like software engineers first and understand what it means to align around the common principles of what it takes to develop safe and secure software that meets its resource and business goals for the domain it serves. A sound software engineer has areas of expertise and skills that include the ability to navigate research and practice tasks as needed. A well-rounded software engineer is equipped to cross great divides of any magnitude, while a researcher or a practitioner by definition is stuck on one side of the divide, often frustrated by not being able to bring the right tools to the task.

References

1. P. Abrahamsson, M. A. Babar, and P. Kruchten, "Agility and architecture: Can they coexist?" *IEEE Softw.*, vol. 27, no. 2, pp. 16–22, 2010, doi: 10.1109/MS.2010.36.
2. S. M. Mitchell and C. B. Seaman, "A comparison of software cost, duration, and quality for waterfall vs. iterative and incremental development: A systematic review," in *Proc. 2009 3rd Int. Symp. Empirical Softw. Eng. Meas.*, pp. 511–515, doi: 10.1109/ESEM.2009.5314228.
3. P. Alexander, "Best of both worlds [formal and semi-formal software engineering]," *IEEE Potentials*,

vol. 14, no. 5, pp. 29–32, Dec. 1995/Jan. 1996, doi: 10.1109/45.481509.

4. V. Garousi, M. Borg, and M. Oivo, "Practical relevance of software engineering research: Synthesizing the community's voice," *Empirical Softw. Eng.*, vol. 25, no. 3, pp. 1687–1754, 2020, doi: 10.1007/s10664-020-09803-0.
5. D. Lo, N. Nagappan, and T. Zimmermann, "How practitioners perceive the relevance of software engineering research," in *Proc. 2015 10th Joint Meeting Found. Softw. Eng.*, pp. 415–425, doi: 10.1145/2786805.2786809.
6. H. Koziolek, "Tracing the practical impact of software architecture research." Medium. https://medium.com/@heiko.koziolek/tracing-the-practical-impact-of-software-architecture-research-a2b91136455 (Accessed: May 12, 2022).
7. C. L. Goues, C. Jaspan, I. Ozkaya, M. Shaw, and K. T. Stolee, "Bridging the gap: From research to practical advice," *IEEE Softw.*, vol. 35, no. 5, pp. 50–57, Sep./Oct. 2018, doi: 10.1109/MS.2018.3571235.
8. "Software Engineering Body of Knowledge (SWEBOK) v3.0," IEEE Computer Society, Washington, DC, USA, 2013. [Online]. Available: https://www.computer.org/education/bodies-of-knowledge/software-engineering

Digital Object Identifier 10.1109/MS.2022.3177873