Trends and Challenges for Software Engineering in the Mobile Domain

Luciano Baresi, Politecnico di Milano

William G. Griswold, University of California, San Diego

Grace A. Lewis, Carnegie Mellon Software Engineering Institute

Marco Autili, University of L'Aquila

Ivano Malavolta, Vrije Universiteit Amsterdam

Christine Julien, University of Texas at Austin

// Mobile computing is becoming a key aspect of our lives. This article builds on conversations held during the 2018 IEEE/ACM International Conference on Mobile Software Engineering and Systems, highlighting mobile computing's impact on software engineering practices, the transition from research to industry, and related areas.//

Digital Object Identifier 10.1109/MS.2020.2994306
Date of current version: 22 December 2020

DURING THE PAST decade, the rapid rise of mobile computing has upended software engineering research and practice.[1] Applications now readily integrate on-device capabilities, from GPS data and wireless communications to cameras and myriad other sensors.[2] They are also immersive, integrating their end users with a quickly changing digital landscape and leveraging dynamically available resources to enable more expressive sensing and reduce energy consumption.

The impact of these changes is evident in both software engineering research and the commercial arena. New sensing contexts, combined with the power of mobility and ubiquity, have radically changed how mobile software is developed in industry settings. The "classical" phases of software engineering no longer apply directly, and novel approaches, frameworks, and the changing landscape demand a substantial shift of perception with respect to these classical approaches. New methodologies, solutions, and frameworks as well as a different way of working are mandatory and require a direct link between research, industry, and end users.

The recent uptick in research directly related to mobile computing has been dramatic and highlights the importance of the topic. At the International Conference on Software Engineering and the Symposium on the Foundations of Software Engineering, the two flagship software engineering conferences, the combined number of articles in the main research tracks relating to "mobile" or "Android" has grown from 14% of all accepted papers in 2016 to 31% in 2018, a 120% increase during the span of two years. On end users' side, the time spent using digital media is strongly driven by mobile devices, with smartphones and tablets accounting for 66%, while

desktop computers represent only 34%.[3] Starting from these figures, this article summarizes the outcomes of a discussion that 65 researchers and practitioners had during a panel session, The Role of Engineering and Development in Mobile Software, held at the 2018 IEEE/Association for Computing Machinery (ACM) International Conference on Mobile Software Engineering and Systems conference in Gothenburg, Sweden.

The panel was designed as a guided conversation about the dimensions shown in Figure 1. The dimensions are orthogonal concerns that crosscut the main stages of the software development life cycle (i.e., requirements, design and development, testing, and maintenance). These dimensions are complementary and give fresh life to the work by Nagappan and Shihab[4] that, at the time it was written (2016), discussed current and future research trends and was organized around those stages.

The Impact of Mobile Computing on Software Engineering Practices

Apps are graphical user interface (GUI)-centered. Efficient, responsive, and appealing user interfaces are key to the success of apps. Apps require ultrafast development cycles because end users rate and review them in tremendous numbers, new versions of Android and iOS are released at least every year, and new competitors continuously pop up in app stores. Apps are very complex software systems that interact with many physical entities (both inside and outside the phone) and integrate diverse (potentially third-party) services in the cloud. Apps must also work on varied devices in terms of the version of the operating system, sensors, and so forth.[5]

Context awareness[6,7] enables the development of innovative features for mobile applications but creates challenges for mobile software engineering. The ability to take advantage of different sensors on mobile devices and to harness data from end users can lead to an improved user experience. However, there are tradeoffs with energy efficiency, security, and privacy that need to be considered as well. We argue that performing tradeoff analysis should be an explicit tenet of the design and development of mobile apps.

Another unique aspect of mobile software development is the need to be adaptive. For example, in power management, nonurgent communications can be delayed to save power (e.g., reporting a user's pulse rate through time), while critical ones should not be deferred (e.g., reporting tachycardia). This points to another unique aspect, which is the constant presence of failure resulting from poor connectivity, low battery levels, sensor inaccuracies, and so on.

These challenges make fully automated approaches to adaptivity difficult. There are times when a (network) failure should not be hidden from end users, who could take action outside an application (e.g., troubleshoot the failure and change the phone's wireless settings). End users desire responsiveness, energy efficiency, always-on connectivity, and, of course, correctness.[8] They also want the latest features. Developers wish for rapid development at reasonable costs. Software researchers have long sought language- and compiler-based solutions for managing tradeoffs. These are critical, but they cannot be a silver bullet.[9]

Research in aspect-oriented software development can find application here. Two recent projects highlight the

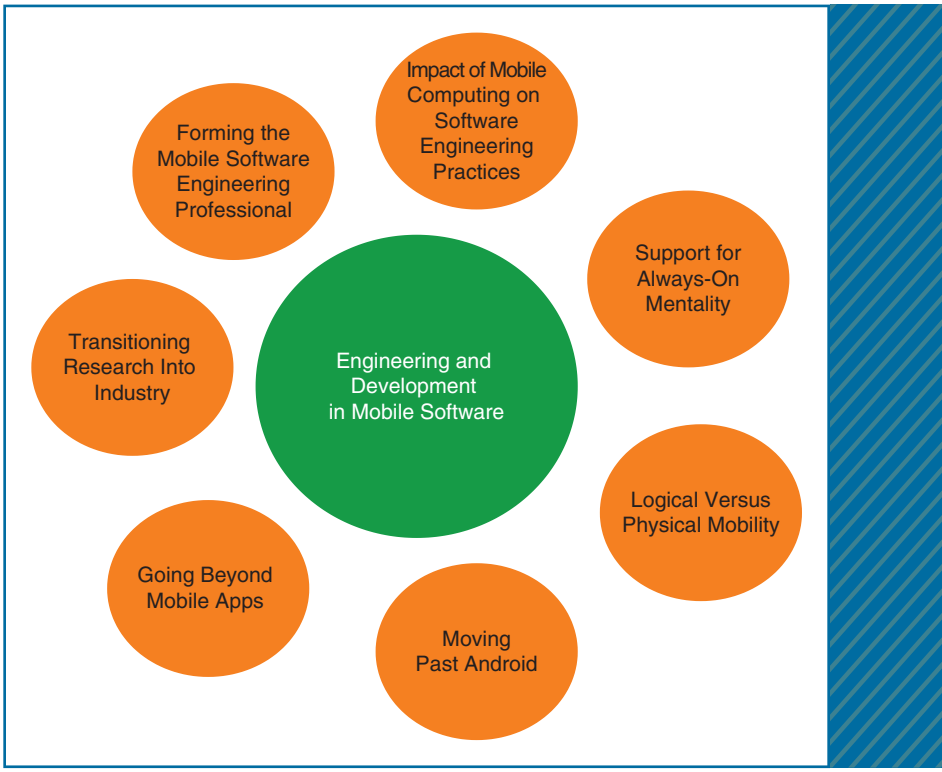

FIGURE 1. The dimensions of the role of engineering and development in mobile software.

promise: APE and ANEL. APE (and its successor, Tempus) addressed the challenges of adaptive power management, enabling a developer to declare which computations, under given situations, could be delayed to save power.[10] ANEL focused on the complexities of robust mobile networking, enabling a developer to incrementally and declaratively improve simple networking implementations.[11] There are also many available libraries, such as Retrofit[12] and Volley,[13] that enable developers to separate complex service consumption and networking code from application and fault-tolerance logic.

The bottom line is that although architecture, system qualities, and tradeoffs have always been a key part of software engineering, addressing these concepts becomes more prevalent and important in mobile software engineering due to much more dynamic operating environments and the diversity of end users and platforms.[4] Providing software engineers with the tools to analyze and study tradeoffs, in addition to resources to help separate concerns, is key for the development of adaptive mobile applications.

Support for the Always-On Mentality

End users want a seamless experience, whether in the home, at work, or on the move. The challenges of supporting their always-on mentality[14] can be addressed only if software engineers maintain a mindset of always-on connectivity. Today's ubiquitous networking pursues the convergence of wireless telecommunication networks and wireless Internet Protocol networks to provide seamless connectivity from everywhere at any time thanks to the multiradio capabilities offered by today's mobile devices. As a result, the ubiquitous networking environment cannot be considered a passive entity that only transports data between end points. Rather, mobile apps must consider the networking environment an active party to be fully exploited. Supporting the always-on mentality in such a complex and dynamic networking environment calls for the development of ubiquitous-oriented solutions, which give mobile apps the "impression" of perfect connectivity, hence assuming the constant accessibility of remote hosts.

> During the past decade, the rapid rise of mobile computing has upended software engineering research and practice.

End users expect their apps to work every time and their devices to always be available. In this respect, defensive programming[15] is a necessary software engineer mentality in today's world. Systems need to be written with the expectation that the worst will happen. Fault tree analysis, a tool commonly used by hardware engineers, is useful for brainstorming about all the bad things that can occur. Chaos Monkey, a project that Netflix developed in 2011 to test its infrastructure, is another example of a tool that reflects this mentality, even though it is not specific to mobile software development. It intentionally disables components in the production network to examine how the remaining systems respond to the outage.

Logical Versus Physical Mobility

Roughly 15 years ago, we distinguished between devices' physical mobility (where mobile devices move around) and their logical mobility (where pieces of code and the state are moved across hosts). We can definitely still differentiate logical and physical mobility, but the real question is whether it makes sense to continue doing so. For the past decade, ever since Mahadev "Satya" Satyanarayanan introduced the concept of cyber foraging,[8] there has been a lot of research examining how to partition and deploy mobile applications so that they can opportunistically take advantage of more powerful computing resources to optimize battery life and network usage. The runtime optimization algorithms that have been developed to determine whether it makes sense to execute locally or remotely are very creative. The techniques to design and partition applications to decide the unit of offload have included offloading threads, methods, classes, services, and full applications.

In the end, the conclusions from all that research have been:

1. It only makes sense to offload if the cost of remote execution is lower than that of local execution.
2. The more information you have to make this decision, the better.

However, if you look at the assumptions made by most of this research, they are impossible to guarantee in

practice. For example, the suppositions include perfect connectivity, flawless knowledge of the information needed in the optimization function, identical local and remote applications that are always accessible, extensive tagging on behalf of the developer, and that there are trusted edge or cloud servers that are always available to receive computation offload requests. The real (and more feasible) power of cyber foraging is in determining how to place data and computation closer to mobile devices at times when smartphones and tablets need them. This means an app is no longer solely responsible for making cyber foraging decisions; instead, the problem becomes a system and network one. Cloud-based systems need to be able to push computation and data to their trusted surrogates, and devices must be able to (automatically) locate these proxies and deal with intermittent connectivity as well as mobility.

Moving Past Android

With 88% of all smartphones running Android, the operating system has a huge portion of today's market share.[16] This is also reflected in the number of scientific contributions to Android at our flagship conferences. For example, of the 105 studies published in the technical track at the 40th International Conference on Software Engineering in 2018, 15 were about mobile apps, and among them, 14 concerned Android. We can also trace this trend toward Android to technical aspects. First, Android is open source, meaning that researchers can customize it by integrating their own solutions and performing experiments on them. Android's source code has been available and versioned since the operating system debuted, enabling researchers to study the internals of the platform and how it evolved through time.

Second, Android is heavily based on Java, a language that is widely adopted and taught in universities. Moreover, many researchers are comfortable with Java, thus easing their understanding of the specific phenomena happening in the source code of Android apps. Finally, a large number of techniques, tools, and data sets exist for statically analyzing and testing Android apps, which can be leveraged by researchers for scaling up the execution of new studies, thereby leaving behind other aspects of the mobile app ecosystem, which are covered less, e.g., in terms of analysis tools.

> End users desire responsiveness, energy efficiency, always-on connectivity, and, of course, correctness.

Nevertheless, if we look back in time, it is evident that the mobile ecosystem is extremely dynamic, with platforms unpredictably rising and falling in terms of device sales (for example, 16% of all smartphones sold in 2010 ran the Blackberry operating system). Company acquisitions and end users flowing from/to other platforms also have an effect. This technological volatility makes us wonder what will happen to the extensive Android-specific body of knowledge and tools that researchers are producing. As a step forward, we suggest that the research community direct its focus to more fundamental aspects of the mobile ecosystem, such as the (currently suboptimal) distribution model of mobile apps, integrated development environments, application programming interfaces provided by mobile platforms, and so on. To make research results more future-proof and relevant, researchers should focus on fundamental challenges in the mobile software engineering ecosystem. As a first step, researchers could commit to have dedicated sections in their papers that concern the generality of the conducted studies, e.g., by showing how they are not specifically bound to Android and by highlighting which parts depend on Android and which can be considered as generally applicable.

Complementary, new languages and cross-platform frameworks are emerging for mobile apps (e.g., Kotlin, React Native, and Flutter). They are extremely popular and heavily adopted in industry, but the research community tends to ignore them. As a group, researchers should get out of their (Java-plus-Android) comfort zone and strive to study new technologies as well as to consider the potential impact that those technologies will have on the mobile computing landscape.

Going Beyond Mobile Apps

Wearables and constellations of Internet of Things devices are revolutionizing the way we live and work (e.g., smart home appliances, Industry 4.0, and health-care advances), and they will drive research in the near future.

New applications of mobile computing will likely have an even bigger impact, such as those enabled by machine learning, computer vision, augmented/virtual reality, natural language processing, and speech recognition. Privacy, security, performance, and energy consumption for mobile apps are being investigated by researchers and practitioners, but they have not been fully explored in the context of these new applications of mobile computing.

From the infrastructure perspective, we are moving toward a continuum where computing starts on mobile devices and continues onto edge and cloud systems, reaching many other devices (e.g., Internet service provider gateways and cellular base stations).[17] We need solutions to exploit such a continuum in a flexible way, without preemptively deciding the allocation of components and thus the languages and tools to create them. Edge infrastructures may add a new, interesting dimension to the problem of managing computationally heavy tasks on mobile devices, especially in the context of wearable devices with very limited battery life and computational power, connected cars, and smart objects (e.g., home automation and advanced logistics).

In this context, a possible step forward may involve combining the microservices architectural style and DevOps concepts for dynamic surrogate provisioning at the edge; protocols that deal better with intermittent communications, such as delay-tolerant networking; and new business models to support cyber foraging on an industrial scale. Cyber foraging will be one of the enablers of augmented reality, given that performing image recognition and analysis are crucial elements of frameworks such as Google's ARCore,[18] but they are extremely expensive for mobile devices in terms of both performance and battery consumption.

> We are moving toward a continuum where computing starts on mobile devices and continues onto edge and cloud systems.

Transitioning Research Into Industry

If we look at research papers about mobile software, we see a spectrum of solutions ranging from narrow studies on low-level aspects of mobile app development (e.g., how Java collections may impact performance) to system-level approaches to a whole ecosystem (e.g., new permission systems for Android). Our perception is that narrow, in-depth studies may be more easily adopted by industry (e.g., a new static checker for Android Studio) compared to wide studies, and their implementation could strongly improve (but also disrupt) the status quo.

Adoptability is also facilitated by the development of well-tested and maintained tools that embody researchers' proposed results, which industry can independently verify and implement. However, on one hand, we know that industry expects ready-to-use solutions, while on the other hand, moving too far toward this model may lead to the risk of not focusing on more fundamental problems, the solutions to which may be more rewarding in the long run. As often happens in software engineering, it is up to the involved players to find the right tradeoff. A first step is to build a certain sensibility with respect to the technology transfer process (using Eshet and Bouwman[7] as starting point) and to learn from success stories (see "Success Stories About Research That Has Impacted the Mobile Industry"). From our experience, we know that when setting up a collaboration, the first and most necessary condition for success is to share and discuss up front the goals of both the academic and industrial partners and allow them to converge. This shields researchers from running the risks of not considering the realities of operational environments and from falling into the trap of trying to sell to industry the answers we have without listening to companies' questions.

The APE approach discussed previously is an example of how the complexities of a research product may be a good fit for a near-term impact. After the APE work was published, Google came out with extensions to the AsyncTask Android class to conditionally run a task based on the network or energy status of a device. Although less powerful than the APE approach, Google's extension was perfect for Android, given the centrality of AsyncTasks in the operating system. In this regard, the APE-related research was less transferable during the near term but more impactful through the long term.

Another problem with some research results is the excessive complexity of the proposed solutions. To address

SUCCESS STORIES ABOUT RESEARCH THAT HAS IMPACTED THE MOBILE INDUSTRY

Notably, two industry sectors, namely 1) medical devices and equipment and 2) network systems and communications, have a recent history of extensive collaboration with academic researchers. For instance, concerning the network systems and communications sector, we can mention the contributions of applied academic research to packet switching and the Transmission Control Protocol/Internet Protocol, which are both key elements in the development of the Internet, routers, asynchronous transfer mode switches, digital subscriber line technology, computer graphics, search engines, traffic management, stable broadcast networking and, last but not least, standards.

At Carnegie Mellon Software Engineering Institute, there are several cases where researchers have started their own companies based on their academic developments. For example, Luis von Ahn, the chief executive officer and cofounder of Duolingo, changed the way that people learn languages. He also founded reCAPTCHA, which was sold to Google in 2009. Fernando de la Torre established FacioMetrics, a company developing technology for facial image analysis that was recently acquired by Facebook. What these ideas had in common was that they were 1) simple; 2) they addressed a very specific need, problem, or idea; and 3) they had a solid implementation of a tool/app to accompany the idea.

These are surely intriguing success stories, but in other cases, the impacts of research can be much more diffused and difficult to track. For example, in 2008 Timothy Sohn (an alumni of the University of California, San Diego) conducted a study on mobile information needs.[S1] His analysis ultimately formed the core of his dissertation and had a relatively high academic impact. But perhaps more importantly, three years later, Sohn was hired at Google and worked on Google Now, which was an early version of the context-aware features in many of the company's mobile apps. It is doubtful that Sohn's study gave Google the idea for Google Now, but his research prepared him to work on the project and helped deliver the product's capabilities to the masses.

Reference

S1. T. Sohn, K. A. Li, W. G. Griswold, and J. D. Hollan, "A diary study of mobile information needs," in *Proc. SIGCHI Conf. Human Factors in Computing Systems*, 2008, pp. 433–442. doi: 10.1145/1357054.1357125.

limitations in APE's path-based power management, TEMPUS takes an object-oriented approach. This requires elaborate static and runtime analyses, which could make TEMPUS difficult for industry to adopt. This kind of complexity was criticized by Willy Zwaenepoel in his 2007 ACM International Conference on Mobile Systems, Applications, and Services keynote, "Peer-to-Peer, Distributed Shared Memory, and Other Products of the Complexity Factory," in which he said, "Complexity is untenable at scale, and scale is what industry does." Researchers and professionals should remember this and focus more on solving problems rather than creating new (and more complex) ones.

Forming the Mobile Software Engineering Professional

When talking about the future of software, we inevitably acknowledge that the supreme value is in students (both of computer science and other disciplines) and their professional education. This leads to the following key question: What is the role of education when forming the next mobile software engineering professional?

Mobile software engineering is a broadly applied topic, which is why a broadly applied program would be preferable. Our hypothetical curriculum would be composed of four main orthogonal dimensions: foundations, experience, business, and research. Foundations is the largest dimension, and it includes software engineering and systems engineering courses at different levels, followed by mobile-specific software engineering lessons. Given that a mobile software engineer is in charge of realizing software that will likely be part of users' everyday life, the experience dimension entails courses related to human–computer interaction, user experience design, GUI design, and so on.

The business dimension aims to provide the instruments to set up and work within a software company (e.g., a start-up) via courses about entrepreneurship and commerce-related topics. As for as the research dimension, students find a real problem to work on, study the current literature regarding solutions to that problem, and propose a solution that is novel and advances the state of the art. Orthogonal to those four dimensions, instructors are in charge of building a rich interface between students and companies to facilitate 1) the students' transition toward positions in industry and 2) the students' exposure to industry-relevant problems and practices.

Lessons Learned and Next Steps

This article summarized the main takeaways that emerged during the panel discussion plus our continued discourse on the topic. Researchers and practitioners can build on the highlighted themes and some lessons learned:

- Diverse sensors and data can lead to an improved user experience, but there are tradeoffs with energy efficiency, security, and privacy.
- Mobile software must be adaptive (e.g., power management) and ready to manage the constant presence of failures (e.g., poor connectivity and low battery levels). Systems need to be written with the expectation that many bad things will happen.
- The research community is urged to focus more on fundamental aspects of the mobile ecosystem instead of being too Android specific.
- We must be ready to manage a continuum where computing starts on mobile devices and continues onto edge and cloud infrastructures.
- Industry expects ready-to-use solutions, but going too far toward that model may lead to the risk of not focusing on more fundamental problems.
- Mobile software engineering is a broadly applied topic that calls for a broadly applied educational programs that include foundations, experience, business, and research.

As for the next steps, there is room to take mobile computing in many different directions:

- *Advancing the mobile experience*: The always-on mentality of today's mobile users calls for the engineering of ubiquitous-oriented network- and middleware-layer solutions that give mobile apps seamless connectivity while guaranteeing security and privacy.
- *Innovation and growth*: The ultrafast, low-latency network capacities and low power consumption promised by 5G call for a new generation of engineers that can revamp the mobile market with cutting-edge apps that fully exploit the power of artificial intelligence-enabled devices coupled with the power of a 5G system.
- *Protection*: Mobile systems are increasingly autonomous in making decisions over and above users or on users' behalf. Often, their autonomy exceeds system boundaries and invades user prerogatives. As a consequence, ethical issues, such as the unauthorized mining and disclosure of personal data and access to restricted resources, are matters of the utmost concern because they impact the moral rights of every human being and affect the social, economic, and political spheres.[19] As a next step, engineers must approach these problems from the regulatory side by contributing to the introduction of new laws and from the technical side by adopting transparency and accountability criteria in software development.

References

1. C. Ebert and K. Shankar, "Industry trends 2017," *IEEE Softw.*, vol. 34, no. 2, pp. 112–116, Mar.–Apr. 2017. doi. 10.1109/MS.2017.55.
2. V. Pejovic and M. Musolesi, "Anticipatory mobile computing: A survey of the state of the art and research challenges," *ACM Comput. Surv.*, vol. 47, no. 3, pp. 1–29, Apr. 2015. doi: 10.1145/2693843.
3. A. Lella and A. Lipsman, "The 2017 U.S. mobile app report," Comscore, Reston, VA, White Paper, 2017.
4. M. Nagappan and E. Shihab, "Future trends in software engineering research for mobile apps," in *Proc. 2016 IEEE 23rd Int. Conf. Software Analysis, Evolution, and Reengineering (SANER)*, pp. 21–32. doi: 10.1109/SANER.2016.88.
5. M. E. Joorabchi, A. Mesbah, and P. Kruchten, "Real challenges in mobile app development," in *Proc. 2013 ACM/IEEE Int. Symp. Empirical Software Engineering and Measurement*, pp. 15–24. doi: 10.1109/ESEM.2013.9.
6. K. Mens, R. Capilla, H. Hartmann, and T. Kropf, "Modeling and managing context-aware systems' variability," *IEEE Softw.*, vol. 34, no. 6, pp. 58–63, Nov.–Dec. 2017. doi: 10.1109/MS.2017.4121225.
7. E. Eshet and H. Bouwman, "Addressing the context of use in mobile computing: A survey on the state of the practice," *Interact. Comput.*, vol. 27, no. 4, pp. 392–412, July 2015. doi: 10.1093/iwc/iwu002.

ABOUT THE AUTHORS

LUCIANO BARESI is a full professor at Politecnico di Milano, Milan, Italy. His research interests include formal approaches to modeling and specification languages, distributed systems, service-based applications, and mobile, self-adaptive, and pervasive software systems. Baresi received a Ph.D. in computer science from Politecnico di Milano, Italy. He is a member of the program committees for multiple conferences and editorial boards of journals in the areas of software engineering, service-oriented computing, and self-adaptive systems, as well as an author and reviewer in these fields. He has also been involved in several European Union and Italian research projects as a technical leader. He was a visiting professor at the University of Oregon, Eugene, and a visiting researcher at the University of Paderborn, Germany. Further information about him can be found at https://baresi.faculty.polimi.it/. Contact him at luciano.baresi@polimi.it.

GRACE A. LEWIS is a principal researcher and lead of the Tactical and Artificial Intelligence-Enabled Systems initiative at the Software Engineering Institute, Carnegie Mellon Software Engineering Institute, Pittsburgh, Pennsylvania, USA. Her research interests include software engineering for artificial intelligence/machine learning systems, Internet of Things security, edge computing, software architecture (in particular, for systems that integrate emerging technologies), and software engineering in society. Lewis received a Ph.D. in computer science from Vrije Universiteit Amsterdam, The Netherlands. She is a member of the organizing and program committees for multiple conferences in the areas of software engineering and software architecture, as well as an author and reviewer in these fields. She is a Senior Member of IEEE and a member of the IEEE Computer Society Board of Governors (2020–2022). Further information about her can be found at http://www.sei.cmu.edu/staff/glewis. Contact her at glewis@sei.cmu.edu.

WILLIAM G. GRISWOLD is a professor of computer science and engineering at the University of California, San Diego, San Diego, California, USA. His research interests include ubiquitous computing, software engineering, and computer science education. Griswold received a Ph.D. in computer science from the University of Washington, Seattle, in 1991. He is a pioneer in software refactoring, and he was a major contributor to the development of aspect-oriented software development. Later, he built ActiveCampus, an early mobile location-aware system. His recent CitiSense and MetaSense projects investigated mobile technologies for low-cost, ubiquitous real-time air-quality sensing. He is a professional member of the Association for Computing Machinery (ACM) and the IEEE Computer Society as well as a past chair of the ACM Special Interest Group on Software Engineering. Further information about him can be found at https://cseweb.ucsd.edu/~wgg/. Contact him at wgg@cs.ucsd.edu.

MARCO AUTILI is an associate professor at the University of L'Aquila, L'Aquila, Italy. His research interests include automated synthesis for composing distributed systems; context-oriented, privacy-aware mobile software programming; resource-oriented analysis of mobile apps; formal specification; and checking temporal properties. Autili received a Ph.D. in computer science from the University of L'Aquila in 2008. He is or has been involved in several European Union and Italian research and development projects as scientific coordinator, scientific and technical leader, research unit coordinator, and work package leader. He is on the editorial boards and program committees of several top-level international journals, international conferences, and workshops. Further information about him can be found at http://people.disim.univaq.it/marco.autili/. Contact him at marco.autili@univaq.it.

ABOUT THE AUTHORS

IVANO MALAVOLTA is an assistant professor in the Department of Computer Science, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands. His research interests include data-driven software engineering, with a special emphasis on software architecture, mobile software development, and robotics software. Malavolta received a Ph.D. in computer science from the University of L'Aquila, Italy, in 2012. He applies empirical methods to assess practices and trends in the field of software engineering. He has authored several scientific articles in international journals and peer-reviewed international conferences proceedings in the software engineering field. He is a Member of IEEE, Association for Computing Machinery, VERSEN, Amsterdam Young Academy, and Amsterdam Data Science. Further information about him can be found at http://www.ivanomalavolta.com. Contact him at i.malavolta@vu.nl.

CHRISTINE JULIEN is a full professor in the Center for Advanced Research in Software Engineering, University of Texas at Austin, Austin, Texas, USA. Her research interests include the intersection of software engineering and dynamic, unpredictable networked environments, with a focus on models, abstractions, tools, and middleware to ease the software engineering burden associated with building applications for pervasive and mobile computing environments. Julien received a D.Sc. from Washington University, Saint Louis, Missouri. Her research has been funded by numerous foundations and organizations. Her work has appeared in many peer-reviewed journal and conference publications, and she regularly serves as a peer reviewer for conferences and journals in these fields. She is a Senior Member of IEEE. Further information about her can be found at http://users.ece.utexas.edu/~julien/. Contact her at c.julien@utexas.edu.

8. G. P. Picco, C. Julien, A. L. Murphy, M. Musolesi, and G. Roman, "Software engineering for mobility: Reflecting on the past, peering into the future," in *Proc. Future of Software Engineering*, 2014, pp. 13–28. doi: 10.1145/2593882.2593884.
9. C. Wohlin et al., "The success factors powering industry–academia collaboration," *IEEE Softw.*, vol. 29, no. 2, pp. 67–73, Mar.–Apr. 2012. doi: 10.1109/MS.2011.92.
10. N. Nikzad, O. Chipara, and W. G. Griswold, "APE: An annotation language and middleware for energy-efficient mobile application development," in *Proc. 36th Int. Conf. Software Engineering*, 2014, pp. 515–526. doi: 10.1145/2568225.2568288.
11. X. Jin, W. G. Griswold, and Y. Zhou, "ANEL: Robust mobile network programming using a declarative language," in *Proc. 5th Int. Conf. Mobile Software Engineering and Systems*, 2018, pp. 202–213. doi: 10.1145/3197231.3197237.
12. Square, "Retrofit: A type-safe HTTP client for Android and Java," GitHub, 2013. [Online]. Available: https://square.github.io/retrofit/
13. Android Developers, "Volley overview," 2020. [Online]. Available: https://developer.android.com/training/volley
14. M. Satyanarayanan, "Pervasive computing: Vision and challenges," *IEEE Pers. Commun.*, vol. 8, no. 4, pp. 10–17, Aug. 2001. doi: 10.1109/98.943998.
15. M. T. Jones, "Defensive programming," Dr. Dobb's, Feb. 1, 2005. [Online]. Available: http://www.drdobbs.com/defensive-programming/184401915
16. Statista, "Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2018," 2018. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/
17. L. Baresi, D. F. Mendonça, M. Garriga, S. Guinea, and G. Quattrocchi, "A unified model for the mobile–edge–cloud continuum," *ACM Trans. Internet Technol.*, vol. 19, no. 2, p. 21, Mar. 2019. doi: 10.1145/3226644.
18. Google Developers, "ARCore," 2020. [Online]. Available: https://developers.google.com/ar/
19. M. Autili, D. D. Ruscio, P. Inverardi, P. Pelliccione, and M. Tivoli, "A software exoskeleton to protect and support citizen's ethics and privacy in the digital world," *IEEE Access*, vol. 7, pp. 62,011–62,021, May 2019. doi: 10.1109/ACCESS.2019.2916203.