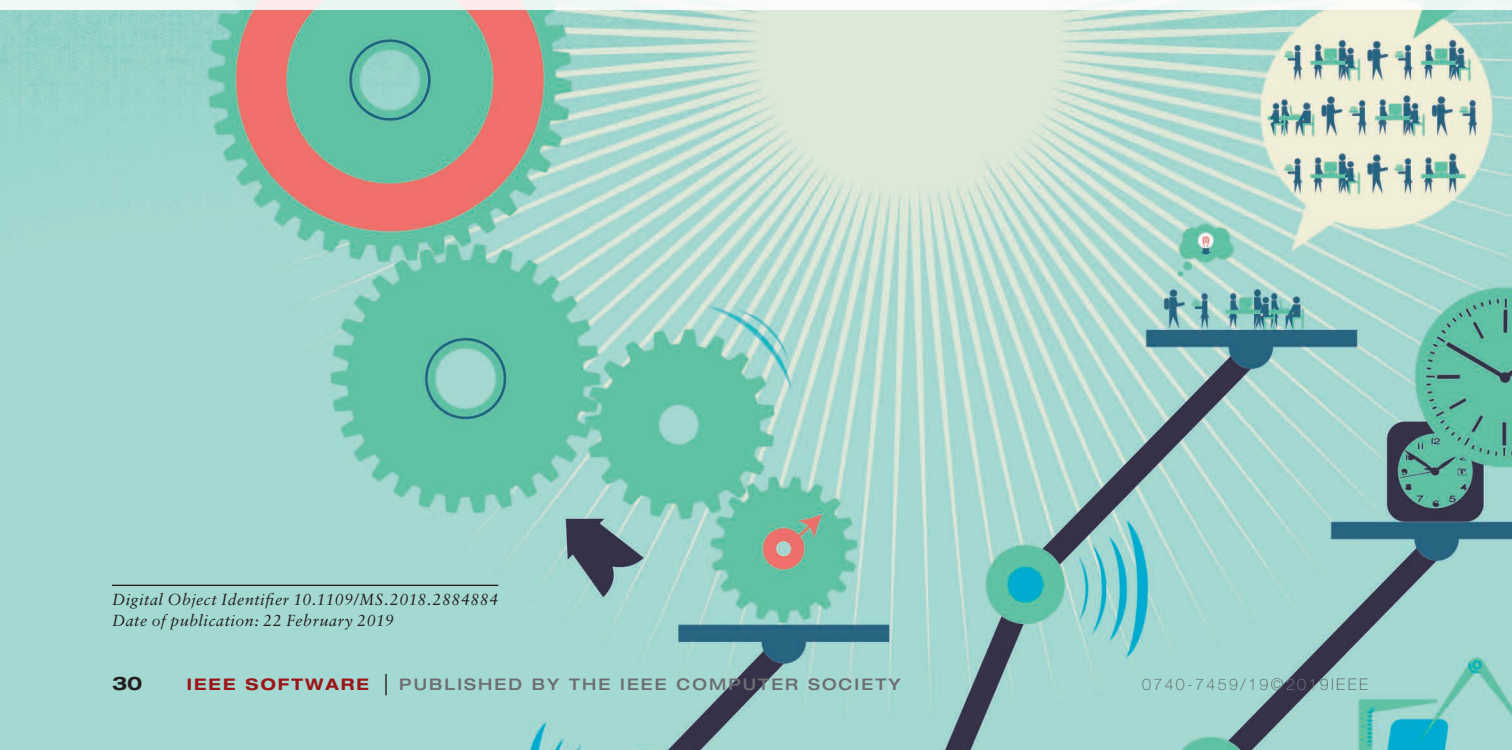
Agile Development at Scale: The Next Frontier

Torgeir Dingsøyr, SINTEF Digital and Norwegian University of Science and Technology

Davide Falessi, California Polytechnic State University

Ken Power



Digital Object Identifier 10.1109/MS.2018.2884884
Date of publication: 22 February 2019

AGILE METHODS HAVE transformed the way software is developed, emphasizing active end-user involvement, tolerance to change, and evolutionary delivery of products. The first special issue on agile development described the methods as focusing on feedback and change.[1] These methods have led to major changes in how software is developed. Scrum is now the most common framework for development in most countries, and other methods such as extreme programming (XP), elements of lean software development, and Kanban are widely used. What started as a bottom-up movement among software practitioners and consultants has been taken up by major international consulting companies who prescribe agile development, particularly for contexts where learning and innovation are key. Agile development methods have attracted interest primarily in software engineering[1, 2] but also in a number of other disciplines including information systems[3] and project management.[4]

Agile software development methods were originally targeted at small, colocated development teams but are increasingly applied in other contexts. They were initially used to develop web systems and internal IT systems but are now used in a range of domains, including mission-critical systems. Methods that were designed for single teams of five to nine developers have been adapted for use in projects with tens of teams and hundreds of developers, which involve integration with hundreds of existing systems and affect hundreds of thousands of users.

Why use agile methods for large projects? Early advice from the agile community was that scaling XP and agile projects is probably the last thing anyone would want to do.[5] Advice from several fields is to reduce the size of software projects; some envision the "death of big software" because new technology allows for microservices, which dramatically reduce the need for coordination.[6] Project management researchers recommend reducing the size of projects[7] to decrease risk, and the general advice from software engineers is to "simplify your product portfolio, reduce the product complexity."[8]

The global focus on digitalization has led to an increased understanding of the importance of software, how it permeates every sector of society, and how it enables competitiveness and innovation.

Although these suggestions favor reducing the size of software projects as much as possible, solutions often require too much work for a single team. This is often because new solutions must be developed quickly, or that new solutions are so complex or so dependent on existing systems that it is deemed inefficient or impractical to split development into small projects. Large telecom products, e.g., typically have more than 20 teams working on the development. Agile methods provide a way to reduce risk at scale while also enabling innovation.

So, what exactly is "large-scale agile development?" A participant in a large project who was being interviewed expressed it as "It is like establishing a medium-size company overnight." The context of that interview was a new project involving a number of external consultants. Many times, large-scale development will be in a product development setting with established teams and established domain knowledge. There will be different needs for the different types of large-scale projects. In "Perform: An Example of a Large-Scale Project," we describe an example of the first type. A common description of large-scale agile development is that of development efforts with more than two development teams, and such projects often have a high number of actors and interfaces with existing systems,[9] which have implications for the development process.

Why is large-scale agile development important now? First, the global focus on digitalization has led to an increased understanding of the importance of software, how it permeates every sector of society, and how it enables competitiveness and innovation. Second, early studies of agile development on a large scale indicate challenges with crucial aspects, such as coordinating teams and work.[10] As new frameworks increase in popularity, more studies are needed. Today, few independent empirical studies exist on how the frameworks work in practice, which circumstances each

PERFORM: AN EXAMPLE OF A LARGE-SCALE PROJECT

The Norwegian Public Service Pension Fund needed a new office automation system because of a public reform. The content of the reform still needed to be passed by parliament after the development had to begin, so the project adopted an agile development method.

Perform [S1] was one of the largest IT projects undertaken in Norway, with a final budget of approximately €140 million. The four-year project comprised 175 people, 100 of whom were external consultants from five companies. Approximately 800,000 person-hours were used to develop roughly 300 epics with approximately 2,500 user stories. These epics were divided into 12 releases.

An existing office automation system was client-/server-based and written in C. The new system was a service-oriented system written in Java. The database from the old system was retained, but the data model was changed. The regulations and legislations were implemented in the new system as rules using JRules. The system was integrated with a new document archive and systems from another public department.

An example release contained the coupling of workflow in the office automation system to an archive solution, a self-service solution for new legislation, simulation of services toward external public departments, and first-data warehouse reports on new data warehouse architecture. Most user stories were identified prior to the first release but were supplemented and reprioritized for every release.

Although it started small, the development project at its peak involved 12 scrum teams working in parallel. There were numerous dependencies among the teams, and to ensure coordination, the teams took on the added roles of technical architect, functional architect, and testing responsibility. They added several extra arenas in addition to Scrum of Scrum meetings. The product was demonstrated every three weeks following the end of an iteration. The product owners were supported by extra resources, with a total of 30 people from the line organization working to define user stories.

The key characteristics were

- 2,300 user stories
- €140 million total cost
- 12 development teams
- 800,000 person-hours
- 12 releases
- 30 people from the line organization involved.

S1. T. Dingsøyr, N. B. Moe, T. E. Fægri, and E. A. Seim, "Exploring software development at the very large-scale: A revelatory case study and research agenda for agile method adaptation," *Empirical Softw. Eng.*, vol. 23, no. 1, pp. 490–520, 2018. [Online]. Available: https:// doi.org/10.1007/s10664-017-9524-2

framework suits best, and what the challenges are and how to overcome them.[11] Finally, company management has become more aware of the importance of software, which leads to a renewed focus on developmental methodologies that ensure competitiveness. The stakes are higher now because these methods are used on larger scales. (See a further detailed discussion about scaling frameworks in "Practitioner Opinion: Agility at Scale: When a Small Cross-Functional Team Is Not Enough.")

For whom is this special issue relevant? It is relevant for decision makers at all levels, whether they are choosing a framework to adopt at the company or project level, tailoring a development model, or selecting key practices for tailoring. The insight provided in this special issue is relevant for managers at software companies, program managers, project managers, facilitators, and developers as well as people in technical roles, product owners, and customer representatives.

This special issue in *IEEE Software* draws upon past studies and experiences, which will complement and sometimes contradict advice from consultants who have developed their own frameworks or models. The methods presented serve a number of functions, from supporting process improvement initiatives with the goal of improving how products and services are delivered (the focus of this issue), to making a company attractive for partners or employees.

PRACTITIONER OPINION: AGILITY AT SCALE: WHEN A SMALL CROSS-FUNCTIONAL TEAM IS NOT ENOUGH

STEVE ADOLPH, SENIOR CONSULTANT
CPRIME

It is a project manager's worst-case scenario: congestive collapse. Everyone is running around "hair-on-fire busy," and nothing is getting done. A looming drop-dead date, which is not just an arbitrary milestone but reflects a real must-ship date, must be met otherwise more than a year's worth of development effort will be lost. Many of the teams are so-called agile teams, but the agile magic is not happening. This is the situation I found myself in at a major industrial equipment manufacturer, with more than 150 engineers trying to fight what they believed was a lost battle. With the clock ticking, we stood the project down for two weeks, trained the teams in a scaling methodology, replanned to create a coordinated backlog, implemented a release plan, and relaunched. The creation of a coordinated program backlog and bringing all of the teams together in a classic big-room planning session aligned the teams and enabled them to focus on getting something done. As a result, a viable product was shipped on time. For this client, the term *agile* was no longer just a team methodology but rather a business strategy.

Nearly 20 years ago, Agile Manifesto captured the imagination of many software developer's burdens with what Alistair Cockburn called the *big-M* methodologies as well as the excesses of the late-1990s software quality rage. Teams began experimenting with agile methodologies and liked the results. Self-reporting surveys repeatedly demonstrated that agile teams created greater customer satisfaction faster, with higher quality and lower costs. Agile demonstrability worked and organizations wanted more of it.

As organizations demanded more, the challenge then became how to create agility beyond the team. Although some teams could create end-to-end value on their own, the team was often just one part of an organization's value-creation process. Also, that team often had to coordinate their work with other teams to create value. Just how did that supposedly omniscient product owner come up with those user stories and what happened to the "increment" after that product owner accepted it? How were the team's fast learning cycles influencing the enterprise's entire value-creation process? How do we coordinate value delivery by multiple teams, and more importantly, how do we coordinate their learning? How do we manage customer needs when a user story only captured a tiny sliver of value; slivers of values so thin, customers often regarded them as merely "nerd" details? Ignoring these questions or simply punting the answers to some higher-level product owner meant that the agile methodologies only provided guidance for small teams or for enterprises where teams could be organized as multiple, independent feature teams.

Some practitioners remembered agile as more than just a basket of methodologies, that it is a strategy, a mind-set. That it is a competitive strategy for creating value by learning faster than the rate of change. That the economics of agility are a function of time and not size. Some agilists began exploring how that mind-set could be applied to larger and more complex systems; realizing agile had not displaced 50 years of software engineering experience. In fact, they looked at how to exploit that knowledge to accelerate the learning process. They began exploring new patterns of planning, such as multilevel adaptive planning, and to integrate the concepts of both intentional and emergent architectures into new patterns of agile architecture. They looked at how to utilize frequent feedback from demos to guide the analysis process, to learn what was truly valuable and to quickly prune less valuable requirements. They sought to balance individuals and interactions ahead of processes and tools at scale.

The so-called scaling frameworks, e.g., scaled agile framework, disciplined agile delivery, Nexus, large-scale scrum, and so on emerged from these patterns. These scaling frameworks integrated patterns for roles, practices, metrics, and supporting artifacts. They addressed the real concerns of practitioners and managers of large and complex systems. Requirements analysis, architecture, design, and long-range planning were all essential. Specialization was sometimes necessary; however, it could be performed in a framework that supported fast learning cycles and adaptation across a number of different time horizons.

(Continued)

PRACTITIONER OPINION: AGILITY AT SCALE: WHEN A SMALL CROSS-FUNCTIONAL TEAM IS NOT ENOUGH (Cont.)

These frameworks accelerated agile adoption among what many had derisively called *laggard* organizations, which were slow to adopt agile because they simply did not believe agilists really understood their needs. Users liked how the scaling frameworks enabled them to see beyond the product owner and offered models for aligning the business with IT. They liked how the scaling frameworks did not just rely on the skills of omniscient product owners or even an omniscient "chief" product owner. They appreciated how the frameworks help create much needed alignment by integrating the fast feedback cycles throughout the whole value-creation process and not just within the software teams. Business and IT began to finally realize they were part of the same organization and had the same goals. Scaling frameworks helped organizations see agility as more than an IT or engineering cost-reduction exercise.

Although fairly intricate, scaling frameworks introduced many new roles, artifacts, and practices. There are even scary reminders of the big-M methodologies. But then product development is complex, and a lack of willingness to understand the context of an organization simply means we will not have credibility with that organization. Simplicity in engineering is good, and Jim Highsmith once described agile methodologies as a *barely sufficient process*. But as Einstein once said, "Make everything as simple as possible, but no simpler." After all, the Agile Manifesto is all about balance.

Agile Development and Scale: The First Iterations

In the context of software development, "agile" was first used in a 1998 *IEEE Software* article[12] to describe in-house methods called *agile software process* and *agile software engineering environment* developed at Fujitsu. Methods such as XP, scrum, Crystal, Evo, and feature-driven development followed the dynamic software development method as iterative development methods.[13] Some credit Microsoft with being the originators of many of the work practices such as continuous build,[11] while some argue that the practices have been common among developers since the 1960s and that agile methods are "old wine in new bottles."[12] Early advice on method tailoring suggested using more disciplined methods when many people were involved.[14]

The first wave of agile methods focused on development in a team setting, with an emphasis on iterative development of high-priority features, prescribing fewer roles, and easy-to-use artifacts that aided the development. Many companies began with long iterations and moved to shorter iterations or continuous development, with many reducing their "ceremony" by using methods such as Kanban. When more development teams were needed, they were coordinated in a separate forum where delegates from participating teams would identify and manage dependencies in tasks among the teams.

Larger development projects would often follow agile methods at the team level, combined with a project management framework such as the Project Management Body of Knowledge or Projects in Controlled Enironments, version 2 (more commonly known as *PRINCE2*). An example of such a project was the 28-month, US$15-million cruise-company project that developed a new web-based customer booking engine.[15] Because of numerous requirement changes, they decided to use agile methods for this project, which was considered successful by its sponsors despite cost and schedule overruns. During the project, roughly 60% of requirements changed, e.g., a requirement that allowed cruise passengers to choose a specific cabin instead of a class of cabins, led to a dramatic change in hospitality practices because previous systems let users book a class and then assign cabins at check-in time. A study conducted of the project describes the combination of structured planning using the

FRAMEWORKS FOR LARGE-SCALE AGILE DEVELOPMENT

Agile portfolio management [S2] is suitable for organizations. Its main characteristics are the following:

- It introduces rolling planning, forecasts, and the dynamic management of portfolios combined with normal agile practices such as standups and retrospectives.
- It establishes a set of core values, including a focus on value, a continuous review of the portfolio, and a demonstration of value from the portfolio.
- It encourages collaboration and empowerment.

Disciplined agile delivery [S3] is suitable for one to many teams. Its main characteristics are the following:

- It is a comprehensive framework that combines ideas from agile development, lean software development, and agile modeling.
- It introduces roles such as stakeholders and architectural owner, specialists, domain experts, technical experts, independent testers, and integrators.
- Twenty new roles appear with scale, such as chief architecture owner, chief product owner, communities of practice lead, and portfolio manager.

The *Kanban method* [S4] is suitable for organizations of any size, from small to very large. Its main characteristics are the following:

- It combines elements of the work of W. Edwards Deming, Eli Goldratt, Peter Drucker, and Taiichi Ohno as well as concepts such as pull systems, queuing theory, and flow.
- It is a significant differentiator from other methods because it starts with where an organization is, and does not require the creation of new roles, ceremonies, or structures before getting started.

Large-scale scrum (*LeSS*) [S5] is suitable for two to seven ("LeSS") or eight-plus ("LeSS-huge") development teams. Its main characteristics are the following:

- A "minimal extension" is required to scrum to handle large-scale product development.
- Product owners are supported by product managers and area product owners in "huge" communities of practice for knowledge sharing and improvement across teams.
- Ideally, most work is done in the feature teams; however, sometimes a separate "undone department" is used to handle architecture, business analysis, quality assurance, or testing.

Nexus [S6] is suitable for three to nine development teams. Its main characteristics are the following:

- Roles, events, artifacts, and rules coordinate the work of approximately three to nine scrum teams working on a single product backlog to build an integrated increment that meets a goal.
- Extra meetings and roles are added on to scrum to coordinate teams.
- It provides a common demonstration for all teams with the same iteration length.

The *scaled agile framework* (SAFe) [S7] is suitable for from groups of 50–150 people to whole organizations. Its main characteristics are the following:

- It is a comprehensive framework that incorporates ideas from agile development and lean production.
- As of version 4.6, there are now multiple versions of SAFe that target organizations and development efforts of different sizes. The versions are branded as essential SAFe, large-solution SAFe, portfolio SAFe, and full SAFe.
- A central concept is the "agile release train," with five to 12 development teams that develop product increments every eight to 12 weeks.

Scrum at scale [S8] is suitable for whole organizations. Its main characteristics are the following:

(Continued)

FRAMEWORKS FOR LARGE-SCALE AGILE DEVELOPMENT (Cont.)

- It focuses on "networks" of scrum teams.
- It separates responsibility for coordinating the "how" and "what" of work in organizations.
- It features a scrum master cycle with advice on how to coordinate scrum teams, and a product owner cycle with advice on coordination of what is to be made (i.e., backlog prioritization).

The *Spotify model* [S9,S10] is suitable for product development with many teams. Its main characteristics are the following:

- It provides a snapshot of a rapidly evolving model.
- It introduced the language of squads (i.e., the basic unit of development), tribes (i.e., the collection of squads working in related areas), chapters (i.e., the people with similar skills), and guilds (i.e., the organic community of interest).
- It features different types of communities of interest or practice established across teams to ensure learning and alignment. It separates architectural roles, such as system owners and a chief architect.

S2. Agile Business Consortium, "AgilePfM," Accessed on: Dec. 18, 2018. [Online]. Available: https://www.agilebusiness.org/agilepfm

S3. The Disciplined Agile (DA) Framework. Accessed on: Dec. 18, 2018. [Online]. Available: www.disciplinedagiledelivery.com

S4. LeanKanban, "What is the Kanban method?" Accessed on: Dec. 18, 2018. [Online]. Available: https://leankanban.com/project/what-is-km/

S5. LeSS. Accessed on: Dec. 18, 2018. [Online]. Available: https://less.works/

S6. Scrum, "The Nexus guide," Accessed on: Dec. 18, 2018. [Online]. Available: https://www.scrum.org/resources/nexus-guide

S7. SAFe, "Welcome to scaled agile framework 4.6," Accessed on: Dec. 18, 2018. [Online]. Available: https://www.scaledagileframework.com/about/

S8. Scrum @ Scale, "Scrum at scale guide," Accessed on: Dec. 18, 2018. [Online]. Available: https://www.scrumatscale.com/scrum-at-scale-guide/

S9. H. Kniberg and A. Ivarsson, " Scaling agile @ Spotify with tribes, squads, chapters & guilds," Spotify, Sweden. Accessed on: Dec. 18, 2018. [Online]. Available: https://blog.crisp.se/wp-content/uploads/2012/11/SpotifyScaling.pdf

S10. Spotify Labs, "Spotify engineering culture (part 1)," Sweden. Accessed on: Dec. 18, 2018. [Online]. Available: https://labs.spotify.com/2014/03/27/spotify-engineering-culture-part-1/

project management framework and the iterative nature of agile methods as enabling the project to learn and adapt.[15]

In a recent historical overview of agile software development, the topic "large-scale agile" appears to originate in the mid-2000s.[16] Eckstein's book *Agile Software Development in the Large*, published in 2004, was the first on the topic. The growing interest in applying agile methods to large projects is illustrated by the ranking of burning research questions by practitioners at the XP Conference, who put "agile in the large" at the top.[17]

A second wave of agile methods sought to address challenges of scale, replacing advice from project management frameworks on addressing layered organizations with portfolios, addressing risks, increasing the number of roles and practices for coordination and alignment across teams, and, in general, picking up on improvement trends, e.g., lean, and adapting them to large-scale software development. Some of the new frameworks provide extensive recommendations for a number of areas such as the scaled agile framework, while others provide less ceremony and recommend more decision authority for autonomous teams, such as in the Spotify model. Thus far, there are few independent studies that show how these new frameworks function, and the trend of making development methods a top-down rather than a bottom-up decision is likely to lead to challenges for adopting new practices. On the other hand, creating awareness among managers of the importance of the development process as well as that continued improvement, is very costly.

In This Issue

This special issue includes contributions on how to address the main

challenges that emerge when using agile methods in large software development projects and programs:

A central question is whether one should apply agile methods to large-scale development. In "Relationships Between Project Size, Agile Practices, and Successful Software Development: Results and Analysis," Jørgensen describes a study of 196 Norwegian IT projects and finds that projects using agile methods outperform nonagile projects when the projects are large. A second question is how to adapt agile methods for large-scale development. We now have a number of new frameworks available; see, e.g., "Frameworks for Large-Scale Agile Development." In "Implementing Large-Scale Agile Frameworks: Challenges and Recommendations," Conboy and Carroll provide recommendations based on a study of 13 agile transformation cases from companies that adopted frameworks such as the scaled agile framework, the Spotify model, and Nexus.

Large-scale agile development will involve numerous people in many development teams. Previous studies have addressed the challenges associated with the lack of alignment among teams.[18] Agile methods primarily rely on oral communication for knowledge sharing through practices such as retrospectives and pair programming. In "Spotify Guilds: How to Succeed With Knowledge Sharing in Large-Scale Agile Organizations," Šmite and her coauthors provide research-based advice on success criteria for what is more commonly known as *communities of practice*.

Another challenge in large-scale development is to ensure customer collaboration because there are often numerous stakeholders. In "Tailoring Product Ownership in Large-Scale Agile Projects: Managing Scale, Distance, and Governance," Bass and Haxby describe the product-owner behaviors that are valued by experienced product owners and line managers, based on their studies of 21 organizations.

A final challenge with scale is decision-making efficiency with large projects or product development efforts. In "Empower Your Agile Organization: Community-Based Decision Making in Large-Scale Agile Development at Ericsson," Paasivaara and Lassenius describe a mode of decision making that seeks to preserve team autonomy in a globally distributed organization with up to 40 teams working on a product.

Today, large-scale agile development is receiving widespread interest. What will happen with respect to the development of methodology advice for large-scale agile development remains to be seen. However, although there is a number of frameworks presented in this article, we believe advice on large-scale agile development is still in a nascent state. An overreliance on frameworks can be a dangerous thing; we have observed evidence of people embracing frameworks for large-scale agile without considering the problem they are trying to solve or whether the framework will really help. Like with agile itself, frameworks should never be the goal; frameworks should help achieve a goal.

Introducing a scaling framework is a significant change for many organizations and should be approached with care. In particular, organizations should consider why something works on a smaller scale before attempting it on a larger scale. There is also the danger of people treating these frameworks as context-free recipes and blindly following the framework without due consideration of their contexts. In the coming years, we need practitioners to share their experiences and express needs for research areas, consultants to pick up research findings, and to continue integrating their experiences from a number of clients. Finally, we need researchers to provide contextually relevant advice by conducting empirical studies and combining lessons with previous research from relevant fields such as project management, organizational psychology, and management science. It is also critical for researchers and practitioners to understand the basic theory behind these practices so that they can better scale them. After all, it is a learning process; scale raises a set of new challenges, but it is still about feedback and change.

Acknowledgments

The work in this special issue was partially supported by the Research Council of Norway under grant 236759. We also thank Knut Rolland at SINTEF and the University of Oslo for comments on an earlier version of this guest editorial. We are very grateful to the authors who have submitted articles and to the reviewers who have contributed greatly to our work.

References

1. L. Williams and A. Cockburn, "Agile software development: It's about feedback and change," *IEEE Computer*, vol. 36, no. 6, pp. 39–43, 2003.
2. T. Dingsøyr, S. Nerur, V. Balijepally, and N. B. Moe, "A decade of agile methodologies: Towards explaining agile software development," *J. Syst. Softw.*, vol. 85, no. 6, pp. 1213–1221, 2012.
3. P. Abrahamsson, K. Conboy, and X. Wang, "Lots done, more to do: The current state of agile systems

ABOUT THE AUTHORS

TORGEIR DINGSØYR is a chief scientist at SINTEF Digital in Trondheim, Norway, and an adjunct professor at the Norwegian University of Science and Technology. His main research interests include software process improvement, teamwork in software development, knowledge management, and large-scale agile development. Contact him at torgeird@sintef.no.

DAVIDE FALESSI is an associate professor of computer science at California Polytechnic State University, San Luis Obispo. His main research interests include devising and empirically assessing scalable solutions for the development of software-intensive systems. Falessi received a Ph.D. in computer engineering from the University of Rome Tor Vergata. He is *IEEE Software's* associate editor for software economics and a Senior Member of the IEEE. Contact him at d.falessi@gmail.com.

KEN POWER has held multiple positions in large technology organizations. His current responsibilities include leading global, large-scale engineering organization transformations. He has been working with agile and lean methods since 1999. He holds patents in virtualization and network management. His main interests include complex adaptive systems, sensemaking, flow-based development, software architecture, distributed systems, artificial intelligence, strategy, engineering management, and leadership. Contact him at ken.power@gmail.com.

development research," *European J. Inform. Syst.*, vol. 18, no. 4, pp. 281–284, 2009.
4. F. Niederman, T. Lechler, and Y. Petit, "A research agenda for extending agile practices in software development and additional task domains," *Project Manag. J.*, vol. 49, pp. 3–17, Oct. 2018.
5. D. J. Reifer, F. Maurer, and H. Erdogmus, "Scaling agile methods," *IEEE Softw.*, vol. 20, no. 4, pp. 12–14, 2003.
6. S. J. Andriole, "The death of big software," *Commun. ACM*, vol. 60, no. 12, pp. 29–32, 2017.
7. B. Flyvbjerg, N. Bruzelius, and W. Rothengatter, *Megaprojects and Risk: An Anatomy of Ambition*. Cambridge, U.K.: Cambridge Univ. Press, 2003.
8. C. Ebert, "50 years of software engineering," *IEEE Softw.*, vol. 35, no. 5, pp. 94–101, 2018.
9. K. H. Rolland, B. Fitzgerald, T. Dingsøyr, and K.-J. Stol, "Problematizing agile in the large: alternative assumptions for large-scale agile development," in *Proc. Int. Conf. Information Systems*, 2016, pp. 1–21.
10. M. Paasivaara, C. Lassenius, and V. T. Heikkila, "Inter-team coordination in large-scale globally distributed scrum: Do scrum-of-scrums really work?" in *Proc. ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, P. Runeson, M. Höst, E.Mendes, A. A. Andrews, and R. Harrison, Eds. Piscataway, N.J.: IEEE Press, 2012, pp. 235–238.
11. C. Ebert and M. Paasivaara, "Scaling agile," *IEEE Softw.*, vol. 34, no. 6, pp. 98–103, 2017.
12. T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Inf. Softw. Technol.*, vol. 50, pp. 833–859, Aug. 2008.
13. C. Larman and V. R. Basili, "Iterative and incremental development: A brief history," *IEEE Computer*, vol. 36, no. 6, pp. 47–56, 2003.
14. B. Boehm and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*. Reading, MA: Addison-Wesley, 2003.
15. D. Batra, W. Xia, D. VanderMeer, and K. Dutta, "Balancing agile and structured development approaches to successfully manage large distributed software projects: A case study from the cruise line industry," *Communications Assoc. Info. Syst.*, vol. 27, no. 1, 379–394, 2010.
16. R. Hoda, N. Salleh, and J. Grundy, "The rise and evolution of agile software development," *IEEE Softw.*, vol. 35, no. 5, pp. 58–63, 2018.
17. S. Freudenberg and H. Sharp, "The top 10 burning research questions from practitioners," *IEEE Softw.*, vol. 27, no. 5, pp. 8–9, 2010.
18. S. Bick, K. Spohrer, R. Hoda, A. Scheerer, and A. Heinzl, "Coordination challenges in large-scale software development: A case study of planning misalignment in hybrid settings," *IEEE Trans. Softw. Eng.*, vol. 44, no. 10, pp. 932–950, 2017.