CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	The future of software development methods
Authors	Fitzgerald, Brian;Stol, Klaas-Jan
Publication date	2017-08-15
Original Citation	Brian Fitzgerald, Klaas-JanStol (2018) 'The Future of Software Development Methods', in Galliers, R. & Stein, M. (eds.), The Routledge Companion to Management Information Systems, London: Routledge, Taylor & Francis Group, pp. 125-137. isbn:9781138666450
Type of publication	Book chapter
Link to publisher's version	http://www.routledge.com/9781317213727
Rights	© 2018 selection and editorial matter, Robert D. Galliers and Maria-Klara Stein; individual chapters, the contributors. This is an Accepted Manuscript of a book chapter published by Routledge in The Routledge Companion to Management Information Systems on 15 August 2017, available online: http://www.routledge.com/9781317213727
Download date	2026-11-02 07:41:20
Item downloaded from	https://hdl.handle.net/10468/6770

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

B Fitzgerald and K Stol (2018) *The Future of Software Development Methods*. RD Galliers and MK Stein (Eds.) The Routledge Companion to Management Information Systems. Routledge. Pp. 125-137

9
THE FUTURE OF SOFTWARE DEVELOPMENT METHODS

Brian Fitzgerald and Klaas-Jan Stol

Introduction

There is a distinct half-life obsolescence with respect to software development methods, in that the principles and tenets that become enshrined in methods can become obsolete due to the emergence of new challenges and issues as the technological environment evolves. This has been labeled the 'problem of tenses' (cf. Fitzgerald 2000; Friedman 1989). The software development methods and processes that are popular today have been derived from principles that were first identified many decades ago – the systems development life cycle, object orientation, agile and lean methods, open source software, software product lines, and software patterns, for example. However, there are a number of fundamentally disruptive technological trends that suggest that we need an 'update of tenses' in relation to the software development methods that can cater to this brave new world of software development. Fitzgerald (2012) has coined the term Software Crisis 2.0 to refer to this new disruptive age. The original software crisis (Software Crisis 1.0, as we might refer to it), identified in the 1960s, referred to the fact that that software took longer to develop and cost more than estimated, and did not work very well when eventually delivered. The diligent efforts of many researchers and practitioners has had a positive outcome in that this initial software crisis has been resolved to the extent that software is one of the success stories of modern life.

Software Crisis 2.0

Notwithstanding the achievements made that have helped to address Software Crisis 1.0, this success has also helped to fuel Software Crisis 2.0. Fitzgerald (2012) identifies a number of 'push factors' and 'pull factors' that combine to cause Software Crisis 2.0. Push factors include advances in hardware such as that perennially afforded by Moore's law, multiprocessor and parallel computing, big memory servers, IBM's Watson platform for cognitive computing, and quantum computing. Also, developments such as the Internet of Things (IoT) and Systems of Systems (SoS) have led to unimaginable amounts of raw data which fuel the field of data analytics – a field that is commonly referred to as "big data." Pull factors include the insatiable appetite of 'digital native' consumers – those who have never known life without computer technology – for new applications to deliver initiatives such as the quantified self, life-logging,

and wearable computing. Also, the increasing role of software is evident in the concept of software defined ⋆ (where ⋆ can refer to networking, infrastructure, data center, or enterprise). The Software Crisis 2.0 bottleneck arises from the inability to produce the volume of software necessary to leverage the absolutely staggering increase in the volume of data being generated in turn allied to the enormous amount of computational power offered by the many hardware devices also available, and both complemented by the demands of the newly emerged digital native consumer in a world where increasingly software is the key enabler. Figure 9.1 summarizes these 'push' and 'pull' factors.

"Software is eating the world!"

This quote from Netscape founder Marc Andreessen colorfully captures the extent to which software is becoming predominant, to the extent that we claim that *all* companies are becoming software companies.

> Our organization has become a software company. The problem is that our engineers haven't realized that yet!

This is how the vice president for research of a major semiconductor manufacturing company, traditionally seen as the classic hardware company, characterized the context in which software solutions were replacing hardware in delivering his company's products. This organization knew precisely the threshold of reuse level for its hardware components before designing for reuse became cost-effective. However, this level of sophistication was not yet present in its software development processes. There was a tendency to approach each software project in a once-off fashion, and a repeatable and formalized software development process had not been fully enacted.

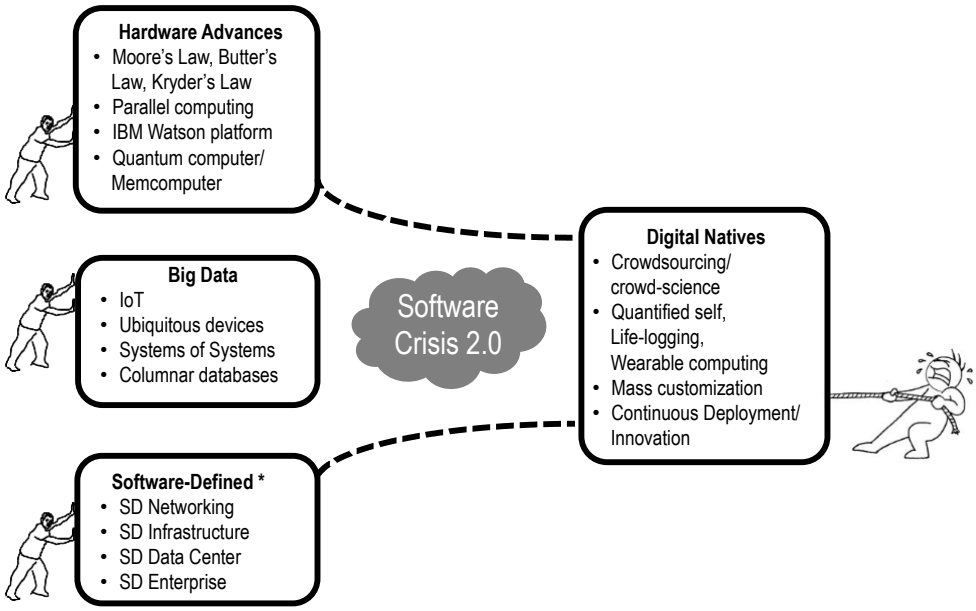


Figure 9.1 Software Crisis 2.0

We have seen this situation replicated across several business domains as the transformation to software has been taking place for quite some time. The telecommunications industry began the move to *softwareization* in the 1970s with the introduction of computerized switches, and currently, the mobile telephony market is heavily software focused. The automotive industry has very noticeably been moving toward softwareization since the 1960s – today, 80%–90% of innovations in the automotive industry are enabled by software (Mossinger 2010; Swedsoft 2010). This is evidenced in the dramatic increase in the numbers of software engineers being employed in proportion to the numbers employed in traditional engineering roles. A striking example of the growing importance of software in the automotive industry is conveyed in the following: In 1978, a printout of the lines of code would have made a stack about twelve centimeters high; by 1995, this was already a stack three meters high; and by 2015, it is a staggering 830 meters, higher than the Burj Khalifa – the tallest man-made structure in the world (see Figure 9.2).

Another example of a domain in which software has increased dramatically in importance is the medical device sector. Traditionally, medical devices were primarily hardware with perhaps some embedded software. However, a 2010 EU Medical Device Directive classifies stand-alone software applications as active medical devices (McHugh et al. 2011). This has major implications for the software development process in many organizations, as they now find themselves subject to regulatory bodies such as the US Food and Drug Administration (FDA).

Thus, we have a context where software, traditionally seen as secondary and a means to an end in many sectors, moves center stage. The implications of this global shift are frequently underestimated. It requires the software development function to transform itself in order to provide the necessary foundation to fulfill this central role, which in turn requires an expansion of the set of concerns that need to be integrated into any software solution. The increasing importance of software is not only a matter of 'scale' in the traditional sense, measured in lines of code, transactions per second, or capacity, but the scaling of software causes the need for changes in other dimensions, too. This has implications for the methods that we use to guide software development practice.

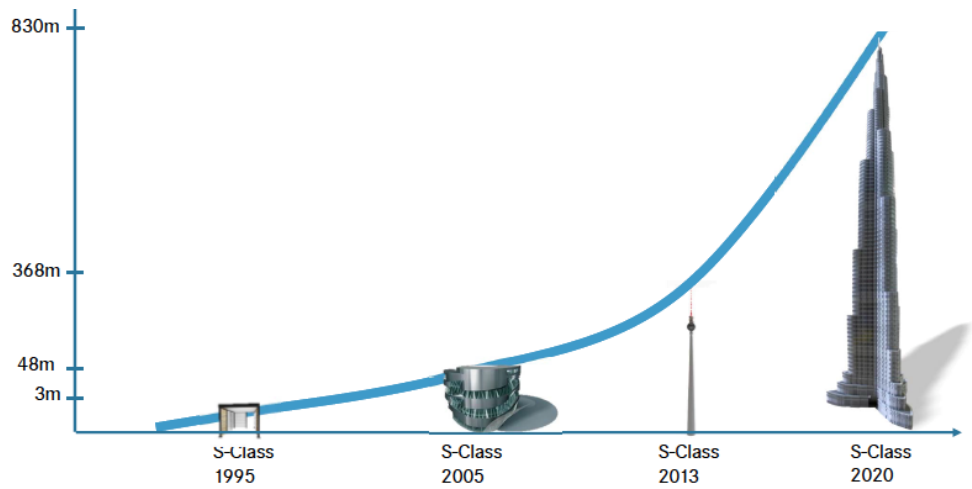


Figure 9.2 Height of software printout stack in Mercedes S-Class

Source: Schneider (2015).

The future of software development methods: an expanded view

This expanded role for the software development function requires a multi-dimensional perspective. Figure 9.3 illustrates three inter-related dimensions that have been drivers for software scaling over the past 20 years. The inner, dark-shaded circle represents the traditional view that was concerned with individual products developed within traditional organizational boundaries using conventional software development processes – or as is the case in many organizations, no formal development process at all. However, the move to services, the emergence of topics such as open innovation, organizational ecosystems, various forms of sourcing (outsourcing, open-sourcing, innersourcing, crowdsourcing; we use the term '⋆-sourcing') (Ågerfalk et al. 2015a; 2015b), and IoT have caused an expanded focus in each of these dimensions. We characterize each of these dimensions in more detail later in this chapter.

Scaling products, systems and services

Companies are also seeking to extend beyond the current state-of-the-art in service-oriented architectures (SOA) to build products. Many companies are actively using open source components and are also providing middleware functionality to provide the glue layer which allows them to incorporate various popular social media applications into their product and service offerings. Software reuse, component-based software development and software product line (SPL) approaches are established means that facilitate the scaling of software systems in the traditional sense.

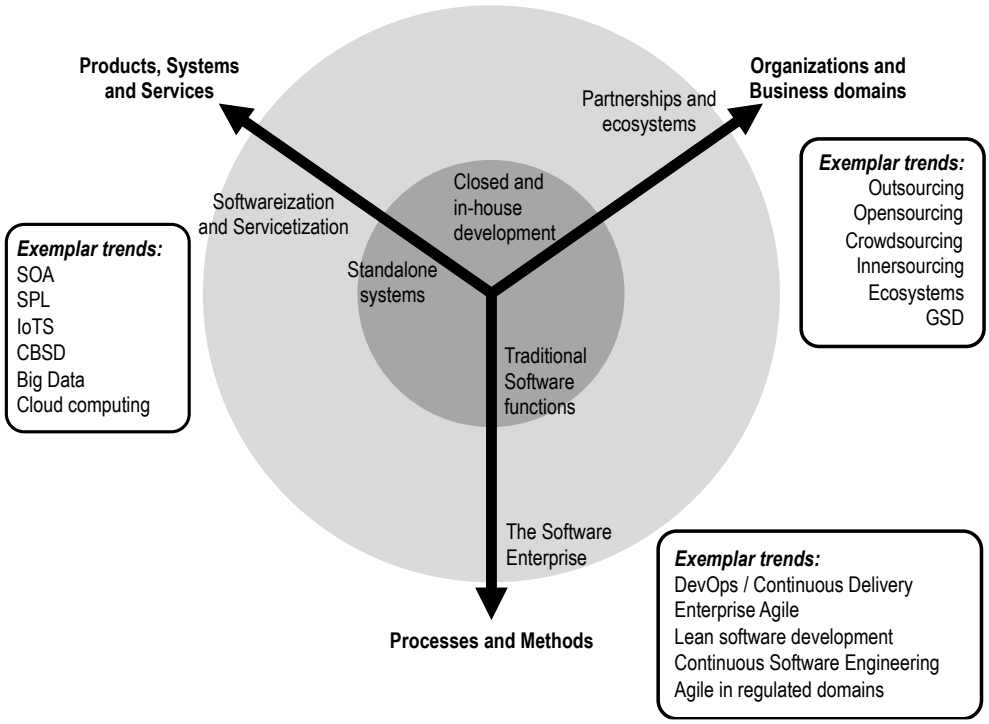


Figure 9.3 Software scaling as a multi-dimensional phenomenon

It is estimated that 35 billion devices are currently connected in the IoT scenario, a figure that is expected to rise to 100 billion by 2020 (Feki et al. 2013). The exponential increase in the volume of available data – Big Data – allied to the ubiquitous devices that are available to process the data also leads to a demand for additional software systems to process into market relevant knowledge. So, not only do software systems become bigger, the number of systems (however small these may be) is growing exponentially.

Another trend is that of *servitization*. Rather than selling products to customers, organizations sell services that could not exist without software. Airbnb is the world's largest supplier of accommodation, but the company owns no real estate. Likewise, Uber is the world's largest taxi company, but owns no taxis of its own. This shift from products to services is continuing and has significant implications for how companies conduct business.

Scaling processes and methods

Software processes and methods are subject to scaling, as building larger systems requires adjusting those processes. New software development paradigms, such as Open Source (Feller et al. 2005), Inner Source (Stol and Fitzgerald 2015), lean software development (Fitzgerald et al. 2014; Poppendieck and Cusumano 2012), and DevOps (Fitzgerald and Stol, 2017), are emerging, and are at varying levels of maturity. Global software development with distributed teams, for example, requires that methods that were not intended for such settings (e.g., agile methods) be tailored. Furthermore, the increasing reliance and prevalence of software solutions in regulated domains (e.g., automotive, medical, and financial sectors, as mentioned earlier) also requires tailoring of both new and traditional development approaches.

While the move from traditional to agile methods is well underway, it is worth noting that agile methods were originally proposed as only suited to small projects with co-located teams developing software in non-critical domains (Boehm 2002). While agile has moved well beyond this space, with frequent use of agile methods on projects with globally distributed teams, many challenges remain in relation to social issues such as cultural differences, communication breakdowns, and optimum practices for distributing development work across sites. Furthermore, the need for Enterprise Agile, or Agile 2.0, whereby agile principles need to permeate beyond the software development function to other organizational functions such as finance, marketing, and human resources, is very apparent. Individual successes such as mentioned earlier rely heavily on their particular organizational context, which is increasingly recognized as an important factor (Dybå et al 2012).

Agile methods and regulated environments are often seen as incommensurable (Turk et al. 2005). The reason for this is probably evident in the Agile Manifesto, which identifies a number of fundamental value propositions for agile – for example, that working software and developer interaction should be viewed as more important than documentation or processes and plans. In regulated environments where traceability is paramount, documentation, processes, and plans are critical. Likewise, in regulated domains, the agile goal of improving time-to-market is secondary to the need to ensure safety is never compromised to satisfy market demands, as the consequences of system failure can lead to loss of life as well as multi-million-dollar compensation claims. Nevertheless, there are encouraging signs to suggest that agile methods can be successfully scaled to regulated environments (Fitzgerald et al. 2013), although this will require organizational changes as well.

Software systems are becoming larger. The obvious expansion is first in terms of lines of code, as demonstrated in Figure 9.2. For example, an average Fortune 100 company maintains 35 million lines of code, adding about 10% each year in enhancements alone – as a result, the amount of code maintained was previously estimated to double every seven years (Muller et al. 1994), but with the ever-increasing ubiquity of computing devices, this is likely to be a conservative estimate. Furthermore, many

organizations are faced with mission-critical legacy software systems that are written in older technologies such as COBOL or even assembly language. This represents major risks for many organizations that wish to modernize their systems. Again, there is a tension in relation to the use of agile methods for such modernization projects. Certainly, the use of agile methods in large development projects remains a significant challenge (Booch 2015), with little empirical evidence of the successful deployment of agile in large projects (e.g., Cao et al. 2004; Kähkönen 2004). The extant literature tends to assume that scaling agile methods can be done in a linear fashion; Rolland et al. (2016) have argued that this approach has limitations and that such assumptions must be re-evaluated.

Scaling organizations and business domains

As software assumes greater importance in companies, traditional concepts such as software architecture, software product lines, and software reuse, formalized software development processes become more important. These in turn require significant organizational change in order to be successfully implemented. A major task here is the education of management and the broader engineering workforce to the complexities and challenges of software development and the benefits of a systematic approach in the areas just mentioned.

Software organizations are now adopting global software development (GSD) strategies such as opensourcing, outsourcing and offshoring, and even crowdsourcing to accommodate the increasing need to deliver larger software systems more quickly in order to stay competitive (Stol and Fitzgerald 2014). Acquisition of companies that possess critical technology or software resources is another common trend. Ensuring a proper governance model and selecting the right partnerships are key challenges. Another trend is that organizations are entering new domains as they increasingly rely on software. Domains that were traditionally dominated by hardware-based solutions are now moving to software solutions. Consequently, companies find themselves in new business markets with new competitors.

Another phenomenon which is relevant in this context is the erosion of strict organizational boundaries as organizations seek to leverage open innovation. One concrete form of open innovation is the building of organizational partnerships or ecosystems with suppliers (e.g., commercial off-the-shelf (COTS) vendors and open source software (OSS) communities) (Bosch and Bosch-Sijtsema 2010). A topic that has started to draw more attention is the creation of open source–style communities within one or a consortium of commercial organizations: (proprietary) software is developed in-house using OSS development practices (Stol and Fitzgerald 2015). This is called Inner Source, and has been adopted in several large organizations such as Allstate, Bloomberg, Bosch, Ericsson, Paypal, Philips, and Sony Mobile. There is growing evidence that an organization's people and culture play a pivotal role in adopting new methods (Conboy et al. 2011). However, there is little insight into how "Communities of Practice" (CoP) can be built within commercial organizations, which must take business considerations and sustainability into account (Tamburri et al. 2013).

Industry vignettes

We believe the three dimensions in Figure 9.3 are inter-related and suggest that to scale successfully in any one dimension requires that related challenges in the other dimensions be resolved simultaneously. This can be difficult as the software development function in many organizations today is not in a position to mandate organizational change in other organizational functions, yet the latter is critical for success. A case in point is the current focus on DevOps and continuous deployment, which requires buy-in from an operations team to be able to deliver new versions at a fast pace (Stol and Fitzgerald 2014).

We illustrate these challenges with a number of industry vignettes describing real-world scenarios where organizations are faced with these challenges. These vignettes represent real organizational

contexts in which we have been involved through funded research projects (Fitzgerald et al. 2013). Each vignette is primarily driven from one dimension but challenges in terms of the other dimensions are clearly evident. The first vignette presents the case of QUMAS and relates to the Scaling Processes and Methods dimension.

Scaling processes and methods: the case of QUMAS

QUMAS is an example of a company developing software for a regulated market. The software development process in QUMAS must adhere to a number of standards. QUMAS previously employed a waterfall-based approach. However, this approach resulted in a long time-to-market and a large release overhead, which were seen as drawbacks in the quickly changing market that QUMAS is operating in. As a consequence, they have adopted and augmented the Scrum methodology.

Agile methods were initially assumed to be limited to small development projects with co-located teams working on non-safety critical development projects. In order to introduce a new agile process, QUMAS have to interact with external organizations in the regulatory bodies that monitor compliance with the relevant standards. Also, the agile process that QUMAS have adopted requires the Quality Assurance (QA) function to assess compliance of the software produced at the end of each three-week sprint. This was a fundamental change in work practice for QA as they currently assess regulatory compliance more or less annually to coincide with new releases of the product. Because QA are required to be independent of the software development function in a regulated environment, senior management need to support such a large organizational change initiative.

QUMAS see this approach as also altering the product and services they offer. New functionality in interim software products at the end of sprints can be demonstrated to customers and their feedback sought more frequently than in the typically annual big-bang release of a new product as per the traditional waterfall-based process.

QUMAS's transformation started on the *processes and methods* dimension, but resulted in significant changes in the product and organizational dimensions. The need to involve other parts of the organization to make a process transformation is clearly relevant to the *organizational* dimension. In regulated domains, the QA function is required to be an independent department from the software development function, and thus the agile transformation also required other parts of the organization to transform. The change to an incremental process facilitated the concept of *continuous compliance*. Whereas the waterfall approach validated the product under development only at the very end, in the new process the compliance auditing took place incrementally as well. The incremental development and frequent delivery also facilitated presales of the product before the software was finished – something unheard of in a waterfall approach. Finally, given the new incremental development approach also affected the implementation of the product, as its design now had to facilitate incremental addition of features.

The second vignette presents the case of Husqvarna, a leading manufacturer of gardening machinery, and represents a company transforming primarily in relation to the Products, Systems and Services dimension, but with major implications for the Organization and Business Domains and the Processes and Methods dimensions.

B Fitzgerald and K Stol (2018) *The Future of Software Development Methods*. RD Galliers and MK Stein (Eds.) The Routledge Companion to Management Information Systems. Routledge. Pp. 125-137

Scaling products, systems and services: the case of Husqvarna

Husqvarna is an example of a company on the journey of transformation to becoming a software company. Ten years ago, Husqvarna was primarily a retailer of gas-powered machinery. Now, an increasing number of machines are battery powered. This changes dramatically the competitive business domain in which Husqvarna operates, as they are now faced with new competitors.

Software is now becoming a key factor. In the past, software development was often done by the robotics staff at Husqvarna, who would have taken programming courses at college. Software development was perceived in Husqvarna as largely equivalent to coding. However, it is estimated that coding only represents about 7% of the effort in software development, and the majority of the work has to do with requirements, architecture and system design, documentation, testing, and other higher-level activities. Husqvarna has moved to the adoption of more formalized software development processes, and a software product line approach is being introduced to rationalize software development across the Husqvarna product range.

A range of new services will be enabled through software in the future. For example, Husqvarna currently incurs a high cost through products being returned as faulty under warranty, which in many cases is due to customer misuse. Adopting an IoT approach, Husqvarna will interrogate built-in sensors in their products deployed in the field worldwide to recover diagnostic and statistical information on the functioning of their equipment. This will enable Husqvarna to proactively advise customers as to whether the equipment needs servicing, and whether it is being used properly and efficiently. If not, a message will be sent to the customer advising them that the product is likely to malfunction, which will require it to be returned to Husqvarna and thus not available to the customer for some period of time.

Also, Husqvarna robotic lawn mowers will be able to monitor weather forecasts to ensure grass is cut before imminent rainfall, and also will remember previous grass-cutting routes to ensure that new routes will be chosen each time. All these innovations are enabled by software.

There are several key lessons in the vignette. First, as Husqvarna was moving from gas-driven to electrically powered machinery, they found themselves in a new business domain that was already inhabited by other companies producing similar devices, and hence were faced with new competitors. In order to introduce new products and services, new approaches such as software product lines and a formalized development process were necessary. Furthermore, the need to deliver more software required an organizational expansion as a result of the hiring of trained software developers.

The final vignette relates to the Scaling Organizations dimension, and we draw on the experiences of Sony Mobile.

Scaling organizations: the case of Sony Mobile

Sony Mobile is an example of a large company in the mobile phone domain, a highly dynamic and competitive market with a number of significant competitors looming. Staying innovative is key and delivering new features is of the utmost importance. This in turn results in a higher demand for a large pool of developers that can deliver those features quickly. To overcome this organizational barrier to scale their software development capacity, Sony Mobile is expanding the development organization in two ways. First Sony Mobile is actively participating and contributing in open source communities. Sony Mobile's phones are based on the Android platform and more than 85% of their software is based on open source. Second, Sony Mobile is adopting Inner Source (adopting open source practices within organizational boundaries). Inner Source facilitates

developers across the organization to collaborate on common projects without formal team membership. Engagement with OSS projects and adopting Inner Source has implications for the product and process dimensions as well. Teams interacting with communities of 'unknown' developers must comply with community norms and common practices so as to ensure that their contributions can be integrated.

Similar to the companies in the other vignettes, Sony Mobile has also been facing the challenge to deliver more software and more quickly. In their domain, they chose to adopt the Android platform for their mobile phones, and consequently were facing the need to collaborate with other stakeholders outside their organization – in this case, open source communities. At the same time, the company is also adopting inner source: managing projects that involve such external and internal communities requires considering new and unknown forces (Höst et al. 2014). Many other large organizations are actively involved in open source communities, including Hewlett-Packard and Wipro. These organizations must learn how to interact with these external communities of developers that may have a different agenda and motivations. To that end, many of these companies are now employing dedicated Open Source Community Experts.

In each of the three vignettes described earlier, the primary focus was on scaling in one dimension, but the companies involved ultimately had to deal with changes in all three dimensions. For example, QUMAS started their scaling transformation in the *process* dimension, but consequentially had to make changes to their *organization* and *product* dimensions. Likewise, Husqvarna is scaling primarily in the *product* dimension, leading to subsequent changes to their *processes* and *organization*. Sony Mobile focused primarily on expanding their *organization* by leveraging open source and inner source communities, which in turn led to changes in the *product* architecture (e.g., dependency on the open source Android platform) and *process* changes.

Implications for research

The three industry vignettes demonstrate how different organizations are scaling in different ways. What they have in common, however, is that no matter which direction they scale in initially, they will eventually have to scale along all three dimensions. This interdependency across these three dimensions has thus far received limited attention; studies tend to focus on one dimension exclusively. For example, cloud computing, software product lines (SPL) and service-oriented architectures (SOA) are all means to scale software products. While the SPL community has also recognized the impact on process and organization, for example through the BAPO framework (van der Linden et al. 2007), there has not been a systematic approach to considering all three interlinked dimensions for all the contemporary developments that we can identify in the modern software research or industry landscapes.

We believe future research on software development should include considerations across all three key dimensions discussed earlier. In Table 9.1, we summarize exemplar scaling scenarios for each dimension, together with the implications for other dimensions, and also some relevant implications for practice, research and education.

B Fitzgerald and K Stol (2018) *The Future of Software Development Methods*. RD Galliers and MK Stein (Eds.) The Routledge Companion to Management Information Systems. Routledge. Pp. 125-137

Table 9.1 Exemplar scaling scenarios in each dimension and their implications

Scaling Dimension	*Exemplar Scaling Scenario*	*Implications for Other Dimensions*	*Implications for Practice, Research and Education*
Products, Systems and Services	Organizations are changing their product offerings and moving to 'softwareization' and 'servitization.'	Companies need to scale on organizational aspects, e.g., hire trained software engineers instead of relying on hardware engineers, and adopt an appropriate software development process that suits the product development context.	The range of software development contexts will become much more varied, which increases the importance of context in research and education of software engineers.
Organizations and Business Domains	Organizations are becoming increasingly dependent on external suppliers and workforces, and offer participation to third-party suppliers in ecosystems.	Trends such as open sourcing, outsourcing and crowdsourcing require changes to an organization's internal development processes. Outsourcing organizations should also consider the product architecture in coordinating external workforces and integrating externally developed software. Keystone players in ecosystems offering platforms to third parties to develop plug-ins or apps must consider constraints of the product architecture.	Organizations must carefully consider the implications of their sourcing and partnership strategies. Software may take an organization to new business domains that are already inhabited by others, thus facing new competition.
Processes and Methods	Organizations are adopting contemporary and emerging software development practices and techniques, including agile methods, DevOps, continuous deployment, and continuous software engineering.	Organizations need to consider other stakeholders in the organization affected by a changing process; e.g., to adopt DevOps and continuous delivery, teams responsible for operations must be involved. Top-level management support is required to get different departments involved. If applicable, constraints due to regulatory compliance must be considered. Product architectures must be amenable to a continuous delivery approach.	Software transcends the software development function, which must be linked to the whole organization – the software enterprise. Software engineering education must address the interactions with other functions, including marketing and sales.

Conclusion

The domain of software development is expanding and reaching far beyond mere technical solutions that involve algorithms, software architectures, and ultra-large systems. The ability to deliver this increasing amount of software is an immediate and pressing concern for many organizations – this has been termed Software Crisis 2.0 (Fitzgerald 2012). Software development must take a holistic view and consider the various interdependencies between trends such as softwareization and servitization, the organizational aspects that are affected by an increasing 'pull' for software-based solutions, and the processes and methods that are used to deliver this software.

These trends will have far-reaching consequences. First, the tension that arises due to the push and pull factors discussed earlier may increase the pressure to deliver software more quickly, which inevitably will lead to compromises in the quality of the software produced. Software that is constructed quickly, without proper design, review, and quality assurance is far more likely to exhibit defects down the road, thus increasing the cost of maintenance, which in turn exacerbates Software Crisis 2.0.

Second, the pull factors also imply an increasing need for talented software developers. As software systems are becoming increasingly complex, the reliance on self-educated and software hobbyists is no longer sufficient; instead, companies need well-trained staff. The lack of well-trained staff is an oft-cited reason for companies to outsource their software development.

Another challenge that is perhaps not receiving enough attention is our society's ever-increasing reliance on software systems. Software systems are truly ubiquitous, but many of these systems are very dated legacy systems. Many critical systems (e.g., operating at banks, insurance companies, and government institutions) were developed in the sixties and seventies on technology platforms that are now considered dated, such as COBOL. While developers who are able to maintain these systems are currently still available, this may no longer be the case 20 or 30 years from now. An incredible amount of software is being written today on a variety of technologies that rapidly become obsolete and will inevitable have to be replaced at some point. As systems are becoming ever more intertwined and interdependent, this will become increasingly problematic.

With the realization that software development transcends algorithms and techniques, we believe research should include considerations across all three key dimensions: products, services, and systems; processes and methods; and organizations and business domains. An increased understanding of the different types of scaling challenges that organizations face will help the software industry to overcome them, and will help in deriving new software development methods better suited to the needs of the prevailing software development environment.

Acknowledgements

This work was supported, in part, by Enterprise Ireland, grant IR/2013/0021 to ITEA-SCALARE, Science Foundation Ireland grant 15/SIRG/3293 and 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero – the Irish Software Research Centre (www.lero.ie).

References

Ågerfalk, P. J., B. Fitzgerald and K. Stol (2015a) *Software Sourcing in the Age of Open: Leveraging the Unknown Workforce*, Springer.

Ågerfalk, P. J., B. Fitzgerald and K. Stol (2015b) Not so shore anymore: The new imperatives when sourcing in the age of open, *Proceedings of the European Conference on Information Systems (ECIS)* Münster, Germany.

Boehm, B. (2002) "Get ready for agile methods, with care," *IEEE Computer*, vol. 35, pp. 64–69; doi:10.1109/2.976920.

Booch, G. (2015) Keynote at the International Conference on Software Engineering, SEIP Track, Florence, Italy.

Bosch, J. and P. Bosch-Sijtsema, "From integration to composition: On the impact of software product lines, global development and ecosystems," *Journal of Systems and Software*, vol. 83, no. 1, 2010, pp. 67–76; doi:10.1016/j.jss.2009.06.051.

Cao, K., P. Mohan, P. Xu and B. Ramesh, (2004) "How extreme does extreme programming have to be? Adapting XP practices to large-scale projects," in *37th Hawaii International Conference on System Science.*

Conboy, K., S. Coyle, X. Wang and M. Pikkarainen, "People over process: Key challenges in agile development," *IEEE Software*, vol. 28, no. 4, 2011, pp. 48–57.

Dybå, T., D.I.K. Sjøberg and D. S. Cruzes, "What works for whom, where, when, and why? On the role of Context in empirical software engineering," *Proceedings of Empirical Software Engineering and Measurement*, 2012, pp. 19–28.

Feki, M. A., F. Kawsar, M. Boussard and L. Trappeniers, "The internet of things: The next technological revolution," *IEEE Computer*, vol. 46, no. 2, 2013, pp. 24–25.

Feller, J., B. Fitzgerald, S. Hissam, and K. Lakhani. (2005) *Perspectives on Free and Open Source Software.* MIT Press. Cambridge, MA.

Fitzgerald, B. "Systems development methodologies: the problem of tenses," *Information Technology & People*, vol. 13, no. 3, 2000, pp. 174–185.

Fitzgerald, B., "Software Crisis 2.0," *IEEE Computer*, vol. 45, no. 4, 2012, pp. 89–91.

Fitzgerald, B., M. Musial and K. Stol, "Evidence-based decision making in lean software project management," in *Proceedings of International Conference on Software Engineering*, vol. 2, Hyderabad, India, 2014.

Fitzgerald, B. and K. Stol, "Continuous software engineering and beyond: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, 2017, pp. 176-189.

Fitzgerald, B., K. Stol, R. O'Sullivan and D. O'Brien, "Scaling agile methods to regulated environments: An industry case study," *Proceedings of International Conference on Software Engineering*, vol. 2, 2013, pp. 863–872.

Friedman, A. (1989) *Computer Systems Development: History, Organisation and Implementation*, Wiley & Sons, Chichester.

Höst, M., K. Stol and A. Orucevic-Alagic (2014) "Inner source project management," in: G. Ruhe and C. Wohlin (Eds.) *Software Project Management in a Changing World*, Springer, pp. 343-367.

Kahkonen, T. (2004) "Agile methods for large organisations – building communities of practice," in *Agile Development Conference.*

McHugh, M., F. McCaffery and V. Casey, "Standalone software as an active medical device," *Proceedings of 11th International SPICE Conference*, 2011, pp. 97–107.

Mössinger, J., "Software in automotive systems," *IEEE Software*, vol. 27, no. 2, 2010, pp. 92–94.

Müller, H., K. Wong, and S. Tilley, "Understanding software systems using reverse engineering technology," *Proceedings of the 62nd Congress of L'Association Canadienne Française pour l'Avancement des Sciences (ACFAS)*, vol. 26, no. 4, 1994, pp. 41–48.

Poppendieck, M. and M. Cusumano, "Lean software development: A tutorial," *IEEE Software*, vol. 29, no. 5, 2012, pp. 26–32.

Rolland, K., B. Fitzgerald, T. Dingsøyr and K. Stol, "Problematizing agile in the large: Alternative assumptions for large-scale agile development," *Proceedings 37th International Conference on Information Systems*, Dublin, Ireland, 2016.

Schneider, J. "Software-innovations as key driver for a green, connected and autonomous mobility," *ARTEMIS-IA/ITEA-Co-Summit*, 2015.

Stol, K. and B. Fitzgerald, "Inner source – adopting open source development practices in organizations: A tutorial," *IEEE Software*, vol. 32, no. 4, 2015.

Stol, K. and B. Fitzgerald (2014) "Two's company, three's a crowd: A case study of crowdsourcing software development," *Proceedings of the 36th International Conference on Software Engineering* (Technical Track), Hyderabad, India

Swedsoft, *A Strategic Research Agenda for the Swedish Software Intensive Industry*, 2010.

Tamburri, D., P. Lago and H. van Vliet, "Uncovering latent social communities in software development," *IEEE Software*, vol. 30, no. 1, 2013, pp. 29–36.

Turk, D., R. France and B. Rumpe, "Assumptions underlying agile software-development processes," *Journal of Database Management*, vol. 16, no. 4, 2005, pp. 62–87.

van der Linden, F., K. Schmid and E. Rommes (2007) *Software Product Lines in Action*, Springer, New York, Inc. Secaucus, NJ, USA.