Reference Architectures and Scrum: Friends or Foes?

Matthias Galster
University of Canterbury
Christchurch, New Zealand
mgalster@ieee.org

Samuil Angelov
Fontys University of Applied Sciences
Eindhoven, The Netherlands
s.angelov@fontys.nl

Silverio Martínez-Fernández
Fraunhofer IESE
Kaiserslautern, Germany
silverio.martinez@iese.fraunhofer.de

Dan Tofan
Independent Researcher
Iasi, Romania
dan.tofan@gmail.com

ABSTRACT

Software reference architectures provide templates and guidelines for designing software systems in a particular domain. Companies use reference architectures to achieve interoperability of (parts of) their software, standardization, and faster development. In contrast to system-specific software architectures that "emerge" during development, reference architectures dictate significant parts of the software design early on. Agile software development frameworks (such as Scrum) acknowledge changing software requirements and the need to adapt the software design accordingly. In this paper, we present lessons learned about how reference architectures interact with Scrum (the most frequently used agile process framework). These lessons are based on observing software development projects in five companies. We found that using reference architectures can be a good agile practice in Scrum: They provide enough design upfront without spending too much effort, reduce documentation activities, facilitate knowledge sharing, and contribute to "architectural thinking" of developers. On the other hand, we found that reference architectures can impose risks or even threats to the success of Scrum (e.g., to self-organizing and motivated teams).

CCS CONCEPTS

•**Software and its engineering** → *Software architectures; Software design engineering; Agile software development;*

KEYWORDS

Software reference architectures, agile software development, Scrum, experience report, lessons learned

ACM Reference format:
Matthias Galster, Samuil Angelov, Silverio Martínez-Fernández, and Dan Tofan. 2017. Reference Architectures and Scrum: Friends or Foes?. In *Proceedings of 2017 11th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, Paderborn, Germany, September 4-8, 2017 (ESEC/FSE'17),* 6 pages.
DOI: https://doi.org/10.1145/3106237.3117773

1 INTRODUCTION

1.1 Context and Background

When it comes to software architecture and design, industry leans towards flexible designs and lightweight architecture design methods [12]. However, even though agile software development argues against big upfront design, "some" upfront architecting is also useful in agile projects [16]. In general, there are two perspectives on architecting software. First, there is the perspective that architectures should emerge and evolve during development. This perspective can be questioned "because architecture encompasses the set of significant decisions about the structure and behavior of the system" [1]. Second, there is the perspective of "big design upfront" which carries the risk of making many architectural assumptions that can be expensive to revert later.

Software reference architectures are one form of upfront design. Choosing a reference architecture is usually one of the first architectural decisions in a project. Reference architectures provide battle-proven (and often generic) design solutions (including generic artifacts, architectural styles, and domain vocabulary) for systems in a particular business domain or technology domain. They include best practices or standards, design guidelines, and sometimes even partial implementations. In this sense, reference architectures are a blueprint for system development and can act as a foundation for designing a concrete (and more refined and detailed) architecture for a particular system. This allows reusing design decisions that worked well in the past [13]. For instance, a reference architecture for web services may describe how a web service is developed and deployed within an IT ecosystem [10].

Prominent examples of reference architectures are AUTOSAR [1] (for automotive software), Oracle's reference architecture for big data systems [2], and vendor-independent reference architectures for web-based systems [3], the Internet-of-Things [17] and OATH for authentication [4]. Software reference architectures come in different flavors depending on their context of use (e.g., used in one or more organizations), origin (either from third parties or designed in-house) and purpose (e.g., facilitating development and standardization) [2]. Typical benefits of reference architectures include higher interoperability of different system components, lower development costs and improved communications within

[1] www.autosar.org/
[2] oracle.com/technetwork/topics/entarch/oracle-wp-big-data-refarch-2019930.pdf
[3] archive.oreilly.com/pub/a/web2/excerpts/web2-architectures/chapter-5.html
[4] openauthentication.org/specifications-technical-resources/

organizations; a drawback of using reference architectures is the additional learning effort required from developers [14].

1.2 Addressed Problem and Contributions

There is an inherent tension between agility in software development and reference architectures. On the one hand, reference architectures, in contrast to concrete system architectures that "emerge" during agile development, constrain the design and development process from the very beginning of a project. Reference architectures dictate parts of the software design by specifying architectural decisions early. On the other hand, agile principles welcome changing requirements. In this paper, we investigate this tension. Our **contributions** are as follows:

- We present lessons learned from using reference architectures with Scrum. These lessons are based on observing projects in five software companies that use Scrum. We focus on Scrum, because it is currently the most widely used agile software development framework [11]. Furthermore, by focusing on Scrum rather than considering agile software development in general, we constrain the context in which our insights are obtained and in which our findings are applicable.
- Based on lessons learned and related observations, we discuss implications for software engineering practice and how the observations can help software developers, designers and team leads use reference architectures in Scrum.

The target audience of our findings are software managers, team leads, developers, designers, architects, and software engineering researchers who would like to understand the use of particular software engineering practices (in this case reference architectures) with Scrum and potential positive and negative effects on project and development agility.

As Bellomo et al. argue, concerns about long-term decrease in the quality of large-scale agile projects have led to an increasing interest of the agile software development community in software architecture [6]. Successfully integrating architecting with agile practices is key to performing architectural work that is useful for agile teams [19]. The role of software architecture and architecting activities in agile development has broadly been discussed in the literature (for example, see Babar et al. [5] and Yang et al. [20] for a comprehensive literature review and Galster et al. [8] for an analysis of the role of the architect in Scrum). Our work complements these works by focusing on the interaction between real-world usages of Scrum (i.e., one particular agile development framework) and reference architectures (i.e., one specific software architecture practice).

2 INDUSTRIAL CONTEXT AND METHOD

To understand the use of reference architectures with Scrum, we observed the use of reference architectures in five global companies.

2.1 Company Profiles

Below we describe the companies involved in our analysis. This allows others to interpret findings by analogy (i.e., our findings may apply to companies which are similar to the companies in this study) [18]. We selected companies based on their use of reference architectures and Scrum. The Scrum framework consists of Scrum teams and their associated roles, events, artifacts, and rules [15]. Therefore, rather than relying on the claims of companies to practice Scrum, we checked if typical roles (Product Owners, Scrum Masters, cross-functional and self-managed Scrum teams), artifacts (product backlog, sprint backlog) and events (sprint planning, stand-ups/daily Scrum, sprint reviews, retrospectives) according to the Scrum Guide (see Schwaber and Sutherland [15]) existed. Company profiles are shown in Table 1.

Table 1: Company profiles

Company	Employees	Domain	Types of software
C1	˜500	Automation	Embedded
C2	˜2,500	Insurance	Web-based
C3	˜41,000	Healthcare	Embedded
C4	˜13,000	Textile	Any
C5	˜20,000	Printing	Embedded

Below we list several commonalities of the five companies:

- Software is developed for external clients rather than for internal use in the companies.
- Companies self-classified their projects as medium to large (based on project duration, budget, amount of code, etc.).
- Typical projects are "greenfield" systems in mature domains. Software products developed are usually novel (e.g., for new products brought to market), rather than evolutions of previously developed products.
- All companies indicated that their software projects have high criticality for achieving business goals of the company.
- Companies indicated that their software must comply with quality standards (e.g., ISO12485) or audit requirements. As a consequence, documentation was a first-class concern in all companies.

In summary, observed companies used reference architectures in stable contexts, for larger systems with low rates of change, and for relatively critical and novel systems.

Reference architectures in companies: Each company used an internally defined reference architecture for its projects. The reference architectures used in the companies include documentation defining basic architecture views, architectural design principles and guidelines that all projects need to follow. Reference architectures used in the companies are not new but have been in use for 3-15 years. For example, C1 defined a reference architecture for Windows-based application development for industrial automation (robotics, automotive, etc.). C2 uses a technology-oriented reference architecture to combine various technologies, such as Java, .NET, Lotus Notes, Visual Basic, etc. C3's reference architecture consists of two documents that describe architectural views as UML diagrams and less formal "boxes and arrows". The reference architecture is described at a high level (in a document that describes mostly design decisions and four major architectural views) and at a lower level (in a document that elaborates on higher level components). Company C4 defined a reference architecture based on their corporate reference model. C5 informally calls its reference architecture "embedded software reference architecture". It consists mostly of documents (more than 20 different documents with 500-1,000 pages) that describe different models, views, component diagrams, sequence diagrams, class diagrams, etc. In all companies, facilitation is the main goal of using reference architectures and standardization is a minor concern only in company C5.

Scrum in companies: All companies have several Scrum teams, depending on the number of projects. All Scrum teams have around five team members, but this might vary per project. In all companies more than one Scrum team might work on a project but one Scrum team usually works on one project at a time.

2.2 Data Collection and Analysis

The primary data source were semi-structured interviews (structured and open questions) with *representative* lead architects and developers in the above companies. Interview questions were centered on a) how reference architectures are used in Scrum projects (e.g., what types of reference architectures, at what stage of a project, who is the driver for using reference architectures), b) what benefits of using reference architectures with Scrum are (e.g., why are reference architectures used and what improved through the use of reference architectures, and c) limitations of using reference architectures with Scrum. We selected one or two representative interviewees per company based on their expertise in using reference architectures and their involvement in Scrum projects in the respective company:

- **C1:** project lead
- **C2:** architect
- **C3:** team lead, architect
- **C4:** architect
- **C5:** team lead, designer

All interviewees had at least an undergraduate degree in software engineering, computer science or related fields. Furthermore, interviewees spent at least five years at organizations in leadership roles and therefore were qualified to provide representative information about companies.

As part of the interviews, we reviewed architecture and process documentation made available by interviewees (however, some information was not shared since companies considered it confidential). We took extensive notes during the interviews, recorded them (with the permission of the interviewees) and transcribed them. To formulate our findings, we followed an inductive approach and inferred findings from coding interviews [18]: We first analyzed how reference architectures impact agile principles and Scrum. Then, we formulated lessons learned explaining these impacts. To clarify and verify our analysis results, we followed up with interviewees in e-mails and phone calls.

3 LESSONS LEARNED

We present seven lessons learned about the impact of using reference architectures in Scrum. In Table 2, we list all lessons learned and map them to companies in which observations related to a lesson were made. Column "Impact" indicates whether a lesson hints at a potential impediment in Scrum caused by reference architectures ('-') or if it hints at aspects of reference architectures that help implement agile principles in Scrum ('+'). Note that this impact is only indicative and we do not make any claims about the strength of that impact.

Table 2: Summary of lessons learned

Lesson	Impact	C1	C2	C3	C4	C5
L1 – Early delivery	+	✓	✓	✓	✓	✓
L2 – Development pace	+	✓	✓	✓	✓	✓
L3 – Documentation efforts	+	✓	✓	✓	✓	✓
L4 – Architectural thinking	+	✓	✓		✓	
L5 – Information sharing	+	✓	✓	✓	✓	✓
L6 – Self-organizing teams	-		✓	✓	✓	✓
L7 – Motivated developers	-	✓		✓	✓	✓

3.1 "Positive" Lessons Learned

3.1.1 Early Delivery of Software. In all companies we studied, reference architectures are used from the very beginning of projects. Companies confirmed that reference architectures provide inspiration for the design of software systems, without putting too many constraints on software development. All pointed out the reduced investment in upfront design *because of* the use of reference architectures since reference architectures already confine the design space and offer proven design decisions for a domain. This reduction in upfront efforts often even outweighs the effort required to learn about a reference architecture. For example, in company C5 the reference architecture helps focus on important design decisions for printing devices at the beginning and avoids producing designs that require heavy redesign later. This allows companies to produce a potentially releasable version of their product earlier than if designs for new software systems were created from scratch.

Lesson 1:
Reference architectures support early delivery of software.

3.1.2 Development Pace. In companies C1 to C5, experiences with using the reference architecture accumulated over time provide reliable data to better plan sprints. In particular, companies C3, C4, and C5 reported simpler effort estimation for user stories. For example, company C3 reported simplified backlog grooming and fewer difficulties when re-estimating incomplete stories during sprint planning. Similarly, company C4 plans the implementation of software modules (their reference architecture advocates module-based systems) in each sprint based on their priority as core or optional modules in the reference architecture. In company C5, teams reuse backlog items that are about instantiating the reference architecture across different projects. Minimizing unexpected or poorly estimated sprint tasks contributes to the ability of teams to follow a constant development pace.

Lesson 2:
Reference architectures support a steady development pace.

3.1.3 Documentation Efforts. Reference architectures in the five companies come with supporting artifacts and documentation, so large parts of concrete system architectures do not need to be documented and maintained separately. This avoids a lot of additional and redundant documentation. Scrum teams and architects focus on documenting product-specific features and design decisions, rather than the whole architecture and all design decisions. Companies that we observed document only deviations from the reference architecture (company C1 documents these deviations in more detail, since their reference architecture is quite generic). For example, as one interviewee in company C3 stated, "we saw a lot of waste about duplication in projects and we reduced that waste by introducing a reference architecture. That is how, at least for us, worked well in more agile software development". The reduction or avoidance of documenting activities allows companies to use working software as a primary measure of progress.

Lesson 3:
Reference architectures reduce documentation efforts.

3.1.4 Architectural Thinking. In companies C1 and C2, reference architectures inject architectural thinking into Scrum. Before using reference architectures, architectural issues were neglected and architectures emerged implicitly. As stated by company C4, reference architectures help "architecturalize" an application. This is particularly useful for Scrum team members without a strong architecting skill set. However, we also found that in companies C3 and C5, architectural thinking and activities are deeply embedded in their software development practices and are not changed by the use of reference architectures. Nevertheless, overall, reference architectures enforce attention to architecting throughout a project allows teams to pay continuous attention to technical excellence and good design.

Lesson 4:
Reference architectures encourage architectural thinking.

3.1.5 Information Sharing. In all companies we found that a shared architectural mindset of core architectural issues helps communicate the shared architectural vision as a "reference point" within Scrum teams during and across sprints. As found in company C1 and C3, this shared understanding of common architectural ideas across different Scrum teams allows engineers to move across different projects and/or teams, and to work on more than one project at the same time (as long as the same reference architecture is used). This facilitates cross-functional teams. Company C5 stated the flexibility of software engineers to switch between projects as a significant benefit since engineers can immediately recognize the structure and terminology in a new project. The existence of a common architectural knowledge facilitates face-to-face communications during projects and therefore impacts positively efficient and effective methods of conveying information to and within development teams.

Lesson 5:
Reference architectures support information sharing.

3.2 "Negative" Lessons Learned

3.2.1 Self-organizing Teams. Self-organizing teams mean that teams have authority to make decisions related to their tasks. In company C1, choosing a reference architecture is a team decision, rather than the decision of the architect only. However, it is the architect (member of the team) who suggests the use of the reference architecture. This means, in company C1, teams are truly empowered and self-organized. In companies C2 to C5, the decision about using a reference architecture is not made by the Scrum team, but externally. In company C2, a software architect (not part of the Scrum team) makes the decision, while in companies C3 and C5 it is lead designers outside the Scrum teams making the decision. In company C4, it is a higher level organizational decision. This shows that despite of empowered and self-organizing teams in Scrum, some decisions are made by leading roles in an organization or project, conflicting with the idea that the best architectures, requirements, and designs emerge from self-organizing teams.

Lesson 6:
Reference architectures may undermine team authority.

3.2.2 Motivated Developers. In companies C1, C3, C4, and C5 some developers feel restricted in their creativity and freedom in making decisions. Furthermore, in company C5, engineers (especially new engineers joining the organization) felt that they need to step out of their comfort zone because of the reference architecture. Some developers complain about the reference architecture as a constraining harness (in a negative way) – "it helps you but it does not feel that nice" (as stated by one interviewee). This discomfort has two reasons: First, developers need to work with (and live with) *major* decisions made by others. Second, developers' decision space and creativity are limited. As recently found, imposed limitations on development related to the technical infrastructure is one of the top-10 causes of unhappiness of developers [9].

Lesson 7:
Reference architectures may frustrate developers in teams.

4 DISCUSSION

4.1 Implications

Overall, our findings contribute to highlighting how software reference architectures work with Scrum. In addition to the positive lessons learned, no company felt that project agility is threatened (unless a reference architecture is used as a strict standardization tool). One may argue that positive lessons learned about reference architectures are about reference architectures in general, no matter if teams follow Scrum or not. However, achieving these benefits directly contributes to allowing Scrum to deliver value. Also, achieving these benefits in Scrum shows that these benefits of reference architectures are not necessarily compromised by using Scrum. Below, we summarize potential implications for practice.

4.1.1 Implications for Scrum Roles.

- **Development team:** Reference architectures can stimulate architectural thinking in Scrum and help developers focus on architectural aspects. This is particularly useful since architecture activities are often performed by engineers from fields with limited education in software architecture [4]. Furthermore, "definitions of done" used by development teams to check whether a user story from the sprint backlog is completed can be lightweight in terms of requirements for documenting related design decisions since reference architectures reduce documentation need.
- **Product Owner:** The Product Owner performs all communications between the team and the other stakeholders (e.g., end users or management roles) outside the team. Product Owners are usually non-technical persons and therefore lack an understanding of the role and meaning of a reference architecture and its impact on Scrum or software projects in general [3]. Typically, Product Owners do not engage in the architecting aspects of a project as they "are often employees of a business department or a different non-IT department" [7]. As reported by Friedrichsen [7], in practice, most Product Owners take care only about the functional requirements but not about the non-functional requirements. Reference architectures therefore allow Product Owners to "ignore" detailed architecting issues but to rely on high-level structures offered by a reference architecture.
- **Scrum Master:** A Scrum Master takes care of the proper implementation of Scrum and related practices. The Scrum Master also helps resolve any impediments that the team may face and manages resources (software, hardware, space, time, etc.). Therefore, understanding risks or threats to the success of the Scrum framework as captured in our lessons learned is essential. Scrum Masters should educate themselves about the reference architecture used in a project, and understand the reasons for using a reference architecture in a particular project. A concrete action of the Scrum Master to support the development team is to help them prepare for the use of the reference architecture at the beginning of a new project (e.g., by arranging trainings for members of the development team).

4.1.2 Risks and Threats to Scrum. Two of our lessons learned indicate negative impacts on self-organizing teams and motivated individuals in Scrum teams. This could be due to the fact that developers may have the perception that deviations from the reference architecture are difficult and therefore inhibits agility and changing requirements (i.e., contradicting a basic agile principle of "welcome changing requirements, even late in development" from the manifesto of agile software development). Furthermore, developers fear that using a reference architecture will introduce some overhead for maintaining and updating the reference architecture (companies in our study used internally defined reference architectures). This may be perceived difficult due to the required speed and focus on creating a potentially shippable release at the end of each sprint, while there is no dedicated time for other activities. This problem may be particularly challenging to deal with if the reference architectures are maintained by architecture bodies (or dedicated teams) outside individual Scrum teams. To mitigate this problem, some team management might be required during project planning or even pre-project work to prepare team members to accept the limitations imposed by reference architectures. To deal with overheads, reference architecture maintenance needs to be planned also with respect to sprint schedules of teams (e.g., maintaining the reference architecture in the middle of a project might cause problems).

4.1.3 Reference Architectures as "Practice" in Scrum. The Scrum Guide states that Scrum "is a framework within which you can employ various processes and techniques." As Bellomo et al. argue, there is a growing consensus that a strong architectural foundation lets teams rapidly evolve complex software systems using iterative, incremental development [6]. Furthermore, Bellomo et al. argue that one major challenge agile teams face in building an architectural foundation is balancing two competing concerns: "delivering near-term functional requirements (based on the agile philosophy of delivering user value early and often) and meeting near- and long-term quality attribute goals (without which the project can grind to a halt because system complexity makes efficient modifications impossible)." This is where reference architectures within Scrum can help. Quality attribute prioritization in particular can be difficult in early increments, and a wrong decision can result in hard-to-modify, unreliable, slow, or insecure systems. Reference architectures can facilitate development by helping developers implement systems meeting particular quality goals based on the context and domain.

4.2 Limitations and Threats to Validity

With regards to construct validity (i.e., did we measure what we intended to measure), our insights are limited since we obtained data from a limited number of sources (interviews and review of architecture and process documentation). However, insights are based on data from different organizations and projects. Also, interviews were based on a common interview guideline and included control questions to check our understanding of questions and answers. With regards to external validity (i.e., are findings of interest beyond the studied companies), we acknowledge that we focus on analytical generalization. This means that our results are generalizable to other companies that have similar characteristics as the ones included in this study. The findings are based on five companies but as can be seen in Table 2, each lesson learned is based on more than one company. With regards to reliability (i.e., how the data analysis depends on those who analyzed the data), we recorded interviews and interview data, and reviewed data collection and analysis procedures before conducting the study. Internal validity is not a concern since our study does not make any claims about causal relationships.

5 CONCLUSIONS

Our study in five companies revealed a mixed impact of using reference architectures with Scrum. In summary, our exploration of the interaction of reference architectures and Scrum helps practitioners mitigate risks and recognize threats in advance. Considering the context of observed organizations and their typical projects, using reference architectures can be a smart approach to architecting activities in projects that develop larger systems in stable contexts. Actionable insights include:

- Reference architectures support several agile software development principles implemented in Scrum, such as continuous delivery and constant development pace.
- Forcing the use of reference architectures on agile teams poses a threat to self-organizing teams and motivated individuals in agile software development projects.

We believe that our findings are applicable beyond the cases presented in this paper as long as the cases have similar contexts as the studied companies. In fact, reflecting on the use of reference architectures can be seen as good "agile practice" based on one of the agile principles in the manifesto of agile software development: "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly."

ACKNOWLEDGMENTS

We thank the interviewees who participated in this study. Also, we thank the reviewers for their insightful comments. This work was partially supported by the ERCIM Fellowship Programme.

REFERENCES

[1] P. Abrahamsson, M. Ali Babar, and P. Kruchten. 2010. Agility and Architecture: Can they Coexist. *IEEE Software* 27, 2 (2010), 16–22.

[2] S. Angelov, P. Grefen, and D. Greefhorst. 2012. A Framework for Analysis and Design of Software Reference Architectures. *Information and Software Technology* 54 (2012), 417–431.

[3] S. Angelov, M. Meesters, and M. Galster. 2016. Architects in Scrum: What Challenges Do They Face?. In *European Conference on Software Architecture (ECSA)*. Springer, 229–237.

[4] P. Oliveira Antonino, A. Morgenstern, and T. Kuhn. 2016. Embedded-Software Architects: It's not Only about the Software. *IEEE Software* 33, 6 (2016), 56–62.

[5] M. Ali Babar, A. Brown, and I. Mistrik. 2013. *Agile Software Architecture – Aligning Agile Processes and Software Architectures.* Morgan Kaufman, Burlington, MA.

[6] S. Bellomo, I. Gorton, and R. Kazman. 2015. Toward Agile Architecture: Insights from 15 Years of ATAM. *IEEE Software* 32, 5 (2015), 38–45.

[7] U. Friedrichsen. 2014. Opportunities, Threats, and Limitations of Emergent Architecture. In *Agile Software Architecture.* Morgan Kaufmann, Boston, 335–355.

[8] M. Galster, S. Angelov, M. Meesters, and P. Diebold. 2016. A Multiple Case Study on the Architect's Role in Scrum. In *International Conference on Product-Focused Software Process Improvement (PROFES).* Springer, 432–447.

[9] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson. 2017. On the Unhappiness of Software Developers. In *21st International Conference on Evaluation and Assessment in Software Engineering (EASE).* ACM, 324–333.

[10] The Open Group. 2011. SOA Reference Architecture Technical Standard. (2011).

[11] VersionOne Inc. 2017. 11th Annual State of Agile Survey. (2017).

[12] M. Keeling. 2015. Lightweight and Flexible - Emerging Trends in Software Architecture from the SATURN Conferences. *IEEE Software* 32, 3 (2015), 7–11.

[13] P. Kruchten. 2004. *The Rational Unified Process: An Introduction.* Addison-Wesley, Boston, MA.

[14] S. Martínez-Fernández, P. Dos Santos, C.Ayala, X. Franch, and G. H. Travassos. 2015. Aggregating Empirical Evidence about the Benefits and Drawbacks of Software Reference Architectures. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).* IEEE, 1–10.

[15] K. Schwaber and J. Sutherland. 2016. The Scrum Guide. (2016).

[16] M. Waterman, J. Noble, and G. Allan. 2015. How Much Up-Front? A Grounded Theory on Agile Architecture. In *37th International Conference on Software Engineering (ICSE).* IEEE, 347–357.

[17] M. Weyrich and C. Ebert. 2016. Reference Architectures for the Internet of Things. *IEEE Software* 33, 1 (2016), 112–116.

[18] R. Wieringa. 2014. *Design Science Methodology for Information Systems and Software Engineering.* Springer Verlag, Berlin/Heidelberg.

[19] E. Woods. 2015. Aligning Architecture Work with Agile Teams. *IEEE Software* 32 (2015), 24–26. Issue 5.

[20] C. Yang, P. Liang, and P. Avgeriou. 2016. A Systematic Mapping Study on the Combination of Software Architecture and Agile Development. *Journal of Systems and Software* 111 (2016), 157–184.