STRATEGIC MANAGEMENT, Vol. 24 (2019), No. 2, pp. 012-033

DOI:10.5937/StraMan1902011D

Received: December 12, 2018
Accepted: March 15, 2019

Agile architecture in the digital era - trends and practices

Zoran Dragičević
Company Boksit, Milići, Republic of Srpska, Bosnia and Herzegovina
Saša Bošnjak
University of Novi Sad, Faculty of Economics in Subotica, Subotica, Serbia

Abstract
The speed of response to change and fluidity are key preconditions for the next generation of IT solutions in the digital world. We are witnessing a rather unimaginable expansion of the use of technology in everyday life, on the one hand, and a continuous increase in the speed of software delivery, on the other, which significantly increased expectations and contributed to the adoption of agile methods and practices, shifting the pendulum of software architecture from traditional to agile methods and practices. Agile architecture, as a result of the transformation of a traditional and agile approach to software development, is a new approach that uses agile techniques to deliver a flexible architecture, adaptable to changing demands, tolerant to changes, which is the result of the iterative-incremental design of the agile process of software development. In recent years, there has been a shift in focus, in practice and research, from people and processes to integration technologies and application's hosting, which has led to the emergence of microservices and increased interest in software architecture and design. One consequence of this is the emergence and development of new approaches in the process of building Agile architecture, such as Continuous Architecting, Lean Architecting or Evolutionary Architecting, which essentially share the same goals. In this connection, in order to understand better the concept and the new role of Agile architecture in the digital era, it is necessary to study the genesis of Agile architecture, as a special approach in software development, to identify current trends and practices that are adapted to the contemporary digital environment (scalability, distribution, complexity). The results of conducted systematic literature review will help researchers and practitioners in better understanding of what Agile architecture is and its role, the current trends and directions of future development, and practices that are particularly useful in the development of complex software, with the aim of broadening the application and improvement of the agile software development process.

Keywords
agile architecture, trends, challenges, success factors, practices, software development

Introduction

The World Economic Forum (WEF) and Accenture launched the *Digital Transformation Initiative* in 2015 to explore the impact of digitalization on business and society. The results released in 2017 predict that the digital transformation will bring a value of $ 100 trillion in the next decade.

Getting cheaper and better technologies, such as mobile, cloud, sensors, analytics and IoT (Internet of Things), with the ability to combine in innovative ways, exponentially accelerate progress. Technology becomes a multiplier in the digital era (WEF & Accenture, 2017) (Figure 1).

The expansion of the technology is accompanied by a continuous increase in the speed of software delivery, which has greatly increased expectations and contributed to the adoption of agile methods and practices. The pillars of software architecture have shifted from traditional to modern methods and practices (Erder & Pureur, 2016), while the speed of response to change and fluidity are key preconditions for the next generation of IT solutions in the digital world (Gartner, 2014).

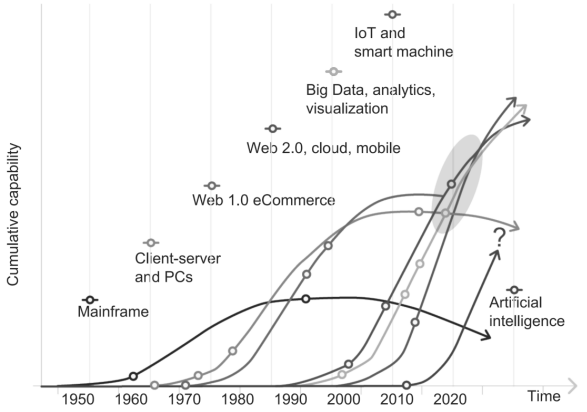

Figure 1 Expansion of technologies in the digital era
Source: WEF & Accenture, 2017.

Agile architecture, as a point of consensus on the process and structure (Kruchten, 2013), is an approach that uses agile techniques to deliver good architecture (Madison, 2010). Kruchten (2013) considers agile architecture as two-dimensional (1) as a software architecture that is versatile, evolving and easily changing, flexible and at the same time resistant to change, and (2) as an agile way to define architecture by an iterative approach, allowing gradual the evolution of architectural design in step with a better understanding of problems and constraints.

Waterman, Noble and Allan (2015) under the term agile architecture mean "*an architecture that satisfies the definition of agility by being able to be easily modified in response to changing requirements, is tolerant of change, and is incrementally and iteratively designed – the product of an agile development process*", where agility is "*the team's ability to create change, respond to change and learn from change so that it can better deliver value*".

With the support of modern practices, technology and tools, the division between the development and production environment is increasingly being blurred, creating a combined ecosystem (Bellomo, Ernst, Nord & Kazman, 2014), so, in the digital age, agile architecture extends to a complete combined ecosystem, which has been particularly influenced by new approaches: Continuous (Erder & Pureur, 2015), Lean (Coplien & Bjørnvig, 2010) and Evolutionary (Ford, Parsons & Kua, 2017) architecture, which add new practices, but essentially share the same goals (Booch, 2010; Holmes & Nicolaescu, 2017).

Through the application of continuous practices, especially continuous delivery, there is a change in focus, in practice and research, from people and processes to integration technologies and application hosting (RESTfull HTTP, cloud computing, DevOps) (Zimmermann, 2016b), which led to the emergence of microservices (Fowler & Lewis, 2014; Newman, 2015) and increased interest in software architecture and design, so that discussion of quality attributes such as scalability, performance, etc. or the discussion of the application of architectural patterns and frameworks, is no longer seen as the BDUF or YAGNI (Zimmermann, 2016b).

Microservices are becoming a key link in the advancement chain that (r)evolutionarily alters the process of software development and delivery (Richardson & Smith, 2016), while new issues arise in the development of complex distributed systems. The digital era brings new challenges that require an innovative approach to finding optimal solutions bearing in mind the wider, combined ecosystem. In this regard, it is necessary to examine in detail current trends and practices of agile architecture in the digital era. Based on the available information there is no study that systematically investigates trends and practices of agile architecture in the digital era.

The rest of this paper is as follows: Section 2 describes the applied methodology, Section 3 presents the results related to the trends and practices of agile architecture in the digital era, Section 4 discusses general considerations, provides answers to research questions, compares similar research, lists identified contradictory attitudes, determines possibilities for further research and constraints. The final part of the paper contains conclusions.

2. Methodology

Research into the trends and practices of agile architecture in the digital era is based on the Systematic Literature Review (SLR) method and the guidelines provided by Kitchenham (2007). SLR is a method for the realization of secondary studies on the results collected from primary studies. SLR protocol was used to define: research question, search process, inclusion/exclusion criteria, quality assessment, method of data collection and analysis. The goal of the research is to answer the following research questions (RQ):

***RQ1**: What are the current trends in Agile architecture, bearing in mind the emergence of different approaches: Continuous, Lean and Evolutionary, in software engineering?*

***RQ2**: What are the practices of developing and implementing Agile architecture in a modern digital environment?*

The search process strategy involved search queries, datasets over which a query and data sources that would be used to identify primary candidate studies (Kitchenham, 2004). Based on research questions, a search query was defined as: *(agile OR lean OR evolutionary OR continuous) AND (architecture OR architecting) AND software AND development.*

In order to increase the probability of finding the desired primary studies, the query was performed on the following datasets: *Title* and *Abstract*. The literature search was carried out by combining automatic and manual searches. The automatic search includes four electronic databases: *IEEE Xplore*, *ACM*, *ArXiv* and *Google Schoolar*, in order to select high quality, reviewed publications in journals and conferences, as well as other publications relevant to the subject of research and research questions.

For each database, based on research questions, a search query is specifically adapted, e.g. for *Google Schoolar* customized query was: *"agile architecture" OR "continuous architecture" OR "lean architecture" OR "evolutionary software architecture"*. A manual search was carried out using the so-called *snowballing* (Jalali & Wohlin, 2012), i.e. iterative searches and finding relevant publications based on references identified in primary publications (*backward snowballing*), as well as publications where primary publications are referenced (*forward snowballing*) (Webster & Watson, 2002).

The inclusion and exclusion criteria were used to assess the suitability of the content of each primary study in relation to the research questions raised (Kitchenham, 2004). The inclusion and exclusion criteria are given in Table 1.

The results of the process of selection of primary studies are presented in Table 2. The selected publications have passed the quality assessment. The quality assessment criteria are defined in the light of the recommendations of Kitchenham (2007) and Dybå and Dingsøyr (2008).

To make an assessment, each publication was subjected to a set of questions that tested the quality of the publication as a whole, the quality, and significance of results and conclusions, as well as the relevance and contribution to the expansion of knowledge and a better understanding of agile architecture. For extraction and qualitative data analysis, the thematic analysis technique and the Atlas.ti software tool was used. The statistics of the selection process regarding the number of primary studies by type of sources and years are shown in Figures 2 and 3.

Table 1 The inclusion and exclusion criteria

Criteria	Assessment Criteria
Include	Publications that define or discuss Agile, Continuous, Lean or Evolutionary Architecture
Include	Publications from journals, conferences, workshop sessions, book chapters, or websites/blogs.
Include	English-language publications, published in the time interval from 2001 to 2017.
Exclude	Publications which are clear that they are not related to the Agile, Continuous, Lean or Evolutionary software development architecture.
Exclude	Publications that just mention the terms Agile, Continuous, Lean or Evolutionary architecture in software development.
Exclude	Non-primary publications (e.g. Systematic Literature Reviews).

Source: Authors

Table 2 Results of the process of selection

Criteria # of articles	IEEE Xplore	ACM	ArXiv	Google Schoolar
After search using query	418	1022	50	145
Selected on title & abstract	47	32	6	25
After removal of duplicates	47	29	6	18
After quality assessment	37	7	1	11
Added by snowballing	5			
Total	**61**			

Source: Authors

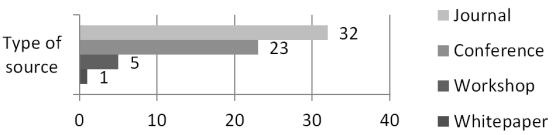

Figure 2 Number of primary studies by type of source
Source: Authors

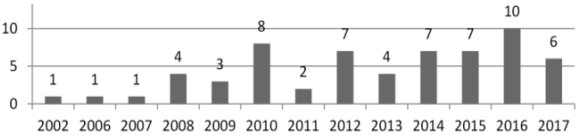

Figure 3 Number of primary studies per year
Source: Authors

3. Results

In this section the results of the research, firstly the trends of the challenges and success factors of agile architecture in the digital era, and then the identified practices, are presented.

3.1. Trends of agile architecture in the digital era

In order to get an answer to the RQ1: *What are the current trends in Agile architecture, bearing in mind the emergence of different approaches: Continuous, Lean and Evolutionary, in software engineering?*, the trends of challenges and success factors have been identified and presented by analyzing selected publications.

3.1.1. The challenges of agile architecture in the digital era

Ten key challenges have been identified, which are associated with several different sources (Table 3).

Table 3 The challenges of agile architecture in the digital era

Challenge	2001-2010	2011-2014	2015-2017
Balancing agility and architecture	S04 S12 S14 S15 S16	S21 S23 S26 S28 S29 S38	S39 S41 S44 S53 S57
Preserving the conceptual integrity and consistency of architecture		S20 S22 S28 S30	S50
Architecture documentation	S03 S09 S18	S30	S42 S47
Scaling	S01 S04 S06 S07 S10	S29 S32	S45 S49 S59 S60
Interdependence of components	S06	S31 S34	S44 S51
Interdependence of requirements	S08	S23 S30	S23 S30
Product life cycle management - optimization of the process and flow of values	S10	S22 S23 S27 S31	S41
Organization, communication and coordination	S06 S07	S28 S29 S31 S32 S35	S42 S45 S47 S49 S53
Application of microservice architecture			S45 S46 S52 S54 S57 S58 S59 S60 S61
New business models of digital economy		S27	S56

Source: Authors

Balancing agility and architecture

Leffingwell, Martens and Zamora (2008) doubt the raising of emergent architecture by refactoring, in the context of the scaled agile process of development and highlight the dangers of over-focusing on urgency (*tyranny of the urgent*). Kruchten (2010) and Abrahamsson, Babar and Kruchten (2010) emphasize the tension between adaptation and anticipation, and the risk of accumulating technical debt as a consequence of an insufficient focus on architecture, while Madison (2010) emphasizes the need to balance business and architectural priorities. For Blair, Watt and Cull (2010), the challenge is to identify the right moment for making key decisions, in order to balance emergent and *Up-Front* design. The issue of improving architectural design in agile methods is being raised (Prause & Durdik, 2012), as well as the issue of optimization of architectural increment in order to achieve the balance of the price of delay of decisions and price of corrections, i.e. refactoring (Nord, Ozkaya & Sangwan, 2012). Waterman, Noble and Allan (2012) explores the design of a minimal up-front architecture, while Fontdevila and Salías (2013) ask the question how to use agile approach and software architecture to increase quality, direct the development process and continuously flow the value for users.

Ozkaya, Gagliardi and Nord (2013) emphasizes importance of integrating agile and architectural principles in order to improve the visibility of project status and to improve risk management tactics when scaling volume, team and/or time. In the context of the increasing importance and wide use of Continuous Delivery (CD), it is particularly important to design architecture for CD (Bellomo et al., 2014).

In the digital era, the pressure is increased for rapid delivery of value (Erder & Pureur, 2016), while maintaining the speed of project realization and product stability (Bellomo, Nord & Ozkaya, 2013), with architecture playing a major role in value streams delivery (Power & Conboy, 2015).

Waterman et al. (2015) puts a focus on the choice of an optimal strategy for the implementation of agile architecture, depending on the degree of influence of different driving forces in a given context. Martini and Bosch (2016) emphasize the challenge of continuous

focus on architecture in order to eliminate technical debt and prevent erosion of architecture because the accumulated technical debt and the degree of refactoring are directly proportional to the misconception about actual architecture (Holmes & Nicolaescu, 2017).

Preserving the conceptual integrity and consistency of architecture

Miyachi (2011) raises the issue of long-term architectural maintenance efficiency, bearing in mind the exponential increase in the cost of correcting errors by the flow of time, especially after the delivery of the software.

Hayata, Han and Beheshti (2012) explores the possibilities of combining lean architecture with agile software development, and the application of lean thinking to preserve the integrity of the architecture. There is a need for a strong focus on preserving conceptual integrity through the planning of regular meetings between top-level design teams (Fontdevila & Salias, 2013), as well as through constant checking of design compliance with architecture (Mirakhorli & Cleland-Huang, 2013).

In the digital era, the importance of long-term preservation of architectural integrity is emphasized, so Erder and Pureur (2016) define an architect as one "*allowing the implementation of software products by directing architectural decisions in a way to protect the conceptual integrity of products*".

Architecture documentation

In order to reduce dependence on undocumented, "*tribal memory*" and preserve intellectual property, Booch (2007; 2010) emphasizes the need for socialization of architecture, whereby it is crucial to find the right measure and method of documenting depending on the complexity of the system. Erdogmus (2009) emphasizes the importance of architecture visibility for making effective decisions, and a particular challenge to documented knowledge is its practical application, as well as the relevance of documentation that is not automatically generated (Mirakhorli & Cleland-Huang, 2013).

In the digital era, the days of presenting architecture with a set of documents are counted, but the architecture is represented by a code executed on the physical infrastructure of the user, the so-called "*realized architecture*" (Erder & Pureur, 2016). Woods (2015) claims that in connection with documentation of architecture, the real challenge is to answer the question "*Who will read it?*", while Gerdes et al. (2016) puts the focus of a challenge of minimal documentation of architecture in order to prevent its erosion, with the requirements: preserve architectural knowledge, improve communication, streamline implementation and support architecture assessment.

Scaling

The intense development of software companies brought challenges to scaling the size of the team (Ambler, 2002; Leffingwell et al., 2008; Moore & Spens, 2008), with a particular problem finding the right people with the desired behavior in large distributed teams (Moore & Spens, 2008). Experience shows that scaling brings challenges in terms of consistency of data and an increase in the number of errors in the race for reaching deadlines (Isham, 2008). Ambler (2009) puts a special focus on the challenge of effectively managing the agile software development process for scaling. Ozkaya et al. (2013) recognizes the challenges associated with three scaling perspectives: the scope, development team and time, while Eckstein (2014) emphasizes the influence of complexity parameters: the degree of change and degree of uncertainty.

In the digital era, one of the major challenges is the scaling of monolithic applications (Villamizar et al., 2015; Taibi, Lenarduzzi, Pahl & Janes, 2017), which often contain a large number of functionality/services, of which a small number requires scaling, causing unnecessary engagement of resources. Scaling also has significant challenges in team organization, communication and collaboration, the role and responsibility of the architect (Britto, Šhmite & Damm, 2016). At the extreme level, scaling brings the greatest challenges that require the use of reactive models for fast response, exceptional elasticity, resistance to failure and asynchronous communication (Pautasso, Zimmermann, Amundsen, Lewis & Josuttis, 2017b).

Interdependence of components

The interdependence of components, including components belonging to a third party (Bellomo, Kruchten, Nord & Ozkaya 2014; Waterman et al., 2015) is a significant challenge that requires increased effort, focus and time (Moore & Spens, 2008), complicates understanding, increases delivery time, and discourages developers to test, implement, and experiment (Buschmann & Henney, 2013).

This challenge is growing in modern complex development and production environment, where systems of systems or software ecosystems are formed, with a multitude of interdependencies between commercial and dedicated software, hardware platforms and organizational entities, each of which has its own evolutionary cycle (Poort, 2016).

Interdependence of requirements

Critical interdependencies in user requirements can lead to significant refactoring with the consequences of the entire structure of the software (Babar, 2009). The hidden interdependencies of the requirements lead to increasing the interdependence of the components (modules) in the design, which further create ad-hoc communication flows between different teams, where teams come to the end of waiting for the completion of the work of other teams or to duplicate the work, causing such conflicts and aggravating management which can ultimately lead to the suspension of the project (Nord et al., 2012). For some extremely complex systems, it is not far from the truth that "*everything affects everything*", and especially the interdependence between functional and non-functional requirements (Mirakhorli & Cleland-Huang, 2013). Poort (2016) considers the interdependence of demands in light of architecturally significant events, adding a time dimension.

Product life cycle management - optimization of the process and flow of values

Although management does not bind to agile approach, agile projects must be managed, because management is directly related to the possibility of scaling, and for this are preferred lean management practices (Ambler, 2009). It is essential to focus on improving workflow and quality while eliminating delays and errors in order to avoid unnecessary work on corrections (Hayata et al., 2012). A particular challenge is an effective management of the workflow process, which consists of linked, interdependent tasks, e.g. when it is necessary to determine the optimal size of architectural increment in order to prevent and eliminate architectural losses (excess production, delays, and defects) (Nord, et al., 2012).

Poppendieck and Cusumano (2012) summarized the aforementioned challenges in a set of principles: optimize the whole, eliminate waste, build quality in, learn constantly, deliver fast, engage everyone, and keep getting better. In this regard, it is necessary to identify and remove any obstacle that frustrates the stakeholders or developers and blocks the work of the team (Buschmann & Henney, 2013). Also, it is necessary to identify the architectural challenges that are obstacles to the flow of value, i.e. the obstacles to the efficient work of the team, for example, unnecessary work, transfer of responsibility, delays, unfulfilled architectural requirements, etc. (Power & Conboy, 2015).

Organization, communication and coordination

When forming teams, it is necessary to find people of appropriate character and behavior (Moore & Spens, 2008) because poor communication between architects and teams working in parallel leads to the problem of "*shooting at a moving target*" (Isham, 2008). Development teams generally do not have enough knowledge or experience to combine agile methods and techniques in the right way, depending on the context (Ramakrishnan, 2010), while, in a scaled context, it is not realistic to require all team members to deal with architecture, because this is sometimes impossible due to size, e.g. 100+ team members (Eckstein, 2014).

Therefore, the need for effective communication between stakeholders (Fontdevila & Salias, 2013; Woods, 2015) and coordination of development teams (Ozkaya et al., 2013; Martini & Bosch, 2016) is emphasized, while ensuring that the development process supports project teams, and not vice versa (Buschmann & Henney, 2013). A particular challenge is to align the architecture and structure of the organization, i.e. vertical and horizontal decomposition of the system, and mapping software modules with people and/or teams responsible for their development, in order to minimize the communication links between teams (Nord, Ozkaya & Kruchten, 2014).

In the digital era, in the conditions of a distributed environment, a close communication is required between the team that developed the module/service and the teams that use it (Villamizar et al., 2015) to avoid problems because of poor mutual understanding due to a different interpretation terms (Gerdes et al., 2016). Poor communication between teams (in relation to communication within teams) leads to conflict in the realization of tasks, which only can be identified by code revision (Britto et al., 2016).

Application of microservice architecture

Non-trivial challenges are inherent in distributed SOA-based systems, such as data integrity, consistency conservation, design and service interface evolution, application/service/ infrastructure management and security (Zimmermann, 2016b). Microservices, as another SOA incarnation, is further characterized by life-cycle challenges (development, testing, delivery, scaling, operations, modification, and replacement), such as: tolerance for failure, distributed transactions, heterogeneous data distribution, versioning, and granularity (Villamizar et al., 2015).

Although microservices and SOA carry great potential to improve flexibility and sustainability by applying and combining their principles and practices, they still need to show long-term cost-effectiveness (Zimmermann, 2016a). Combined development and production environment is complicated, as microservices require a sophisticated DevOps infrastructure, based on cloud and container technologies, which support a hyper-agile, lean process of software development and delivery (O'Connor, Elger & Clarke, 2017).

The application of microservices may be difficult in conditions of tightly connected components. With the challenge of finding a balance between complexity and flexibility, security and other quality attributes problems are possible (Holmes & Nicolaescu, 2017), with a particular emphasis on finding balance between performance and granularity (Shadija, Rezai & Hill, 2017). Many developers have a problem due to the change of paradigm from *in-process* to *calls across* a process boundary, as well as problems with versioning and error management, while redundancy in the implementation of microservices (coarse-grained & fine-grained) often prevents reuse (Pautasso et al., 2017a).

The challenge may also be the coordination of the work around the API gateway, i.e. the appearance of a bottleneck that blocks other teams. In addition, it is important to automate testing and monitoring to answer the questions of how to find out something is wrong and how to collect data for the purpose of visualization. It is essential to understand the fact that a large system has different rules from a small system, e.g. in a small system, redundancy is avoided, transactions are processed and a common data model is defined, while in a large systems redundancy is required, compensation is used instead of transactions, and a common data model is a recipe for failure. With this in mind, it is ***particularly problematic that programming is mainly taught on small systems***. This situation requires greater use of tools, technologies, and designs at the system level to hide delivery and scaling jobs from developers (Pautasso et al., 2017b).

New business models of digital economy

The software value-creation epicenter has changed, so instead of focusing on transaction management and equipment control, new business models, such as *Two-sided Market* and *Creating engaging experiences,* require the construction of ecosystems that will attract users with the ability to understand and focus on the important needs of users who are not adequately serviced. The user experience design is the fundamental element of this approach (Poppendieck & Cusumano, 2012).

In the digital era, wide access to the Internet, mobile devices, SaaS (*Software-as-a-Service*) products, massively consumed startup products have led to a change from B2B to B2C and the emergence of a *pay-per-use* business model (Villamizar et al., 2015). The API economy further increases the complexity, so modern systems are expected to be automatically, horizontally scaled to the required number of machines, automatically delivered anywhere, manageable, exchangeable, resistant, with zero tolerance for failure, self-adjusting and unbreakable. The boundaries of the context of Internet-scaled systems and their architecture become blurred, while monolithic software applications have been changed by Internet-based ecosystems based on microservice architecture with more dynamic and complex runtime characteristics (Hohpe, Ozkaya, Zdun & Zimmermann, 2016).

3.1.2. The success factors of agile architecture in the digital era

Ten key success factors have been identified which are associated with several different sources (Table 4).

Understanding the context and selecting the implementation strategy

One of the key characteristics of the agile process is a strong awareness of the context, i.e. the ability to know what is going on (Madni, 2008), while Babar (2009) argues that in agile approach context analysis, definition of problems and specifications of the request are shifted to the user.

Table 4 The success factors of agile architecture in the digital era

Success factor	2001-2010	2011-2014	2015-2017
Understanding the context and selecting the implementation strategy	S05 S08 S16 S17	S21 S26 S35 S37	S41 S44 S46 S50 S56 S58 S59
Understanding the role, responsibilities and competencies of an architect	S04 S08 S13 S15	S19 S24 S25 S30 S31 S32 S35 S36	S42 S46 S49 S50 S53 S56 S57 S58
Traditionalization of agile approach	S03 S09 S18	S30	S42 S47
Application of lean principles and practices	S01 S04 S06 S07 S10	S29 S32	S45 S49 S59 S60
Application of continuous principles and practices	S06	S31 S34	S44 S51
Use of architectural styles, design patterns and components	S08	S23 S30	S23 S30
Decomposition and granularity	S10	S22 S23 S27 S31	S41
An evolutionary approach	S06 S07	S28 S29 S31 S32 S35	S42 S45 S47 S49 S53
Quality attributes - continuous focus and prioritization			S45 S46 S52 S54 S57 S58 S59 S60 S61
Application of tools and technologies in the combined ecosystem		S27	S56

Source: Authors

For Abrahamson (2010) the context is crucial for balancing agility and architecture, whereby the context includes: project size, architecture stability, business model, team distribution, degree of change, age of the system, criticality and management, as well as the influence of other factors: the market situation, the strength and policy of the company, the expected life cycle of the product, the type of product, organizational culture and history.

Although the software architecture is relevant to the members of the agile teams (Falessi et al., 2010), agile methods are only suitable for projects in a particular context (Prause & Durdik, 2012), while the context determines the amount of *Up-Front* work on architecture (Waterman et al., 2012). Nord et al. (2014) emphasize the importance of the context for aligning system architecture, organization structure and production infrastructure, while 20 contextual factors related to project, team, practices and organization determine whether architecture can emerge as a result of continuous refactoring (Chen & Babar, 2014) .

In the digital era, context change is one of the main obstacles to the continuous process flow (*workflow*) (Power & Conboy, 2015). Architects, depending on the particular context, must adapt *Risk-and-Cost Driven Design* (RCDD) methods, pragmatic modeling, and technical debt management to make effective decisions (Zimmermann, 2016a), so organizations values the architect by ability to make the right decisions in an unclear context (Erder & Pureur, 2016). Internet-connected architecture adds complexity and blurs the boundaries of the system context

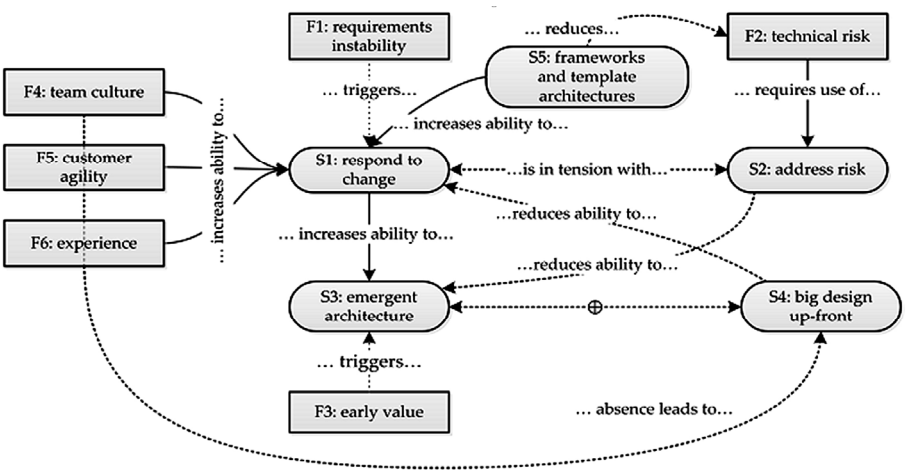

Figure 4 Agile architecture forces and implementation strategies
Source: Waterman et al., 2015.

(Hohpe et al., 2016). *Domain Driven Design* (DDD), bounded context and *Conway's law* are key success factors of microservices (Pautasso et al., 2017a), while changing boundaries of the service context over time is one of the most significant challenges of microservice architecture (Pautasso et al., 2017b).

Based on the importance of the influence of individual driving forces (demand instability, technical risk, rapid delivery of value, team culture, user agility and experience) in a given context, one of the following strategies for implementing agile architecture can be selected (Figure 4) or combine multiple strategies (Waterman et al., 2015):

Respond to changes - a strategy directly linked to the agility of teams where greater agility of architecture and new tolerance for change allows architecture to continuously present the best solutions to the evolving problem. The team can use the following tactics: simple design, iterative proactive architecture revision, the use of good design practices, delaying architectural decisions, and planning for options.

Addressing risk - reduces the impact of risk before the problem arises, usually *Up-Front*, especially for decisions that have a wide impact (e.g. selection of technological streak or architectural style), where architecture is designed so that it is possible to build a system with the required quality attributes with an acceptable level of risk.

Emergent design - the team adopts minimum *Up-Front* decisions, such as the choice of technology stack or architectural style/pattern, whereby these decisions are sometimes implicit, or have already been made (e.g. by users), and can be viewed as constraints, then we have the so-called total emergent design. The team considers only current requirements, ignoring long-term, with the simpler design that allows the product to reach the market as soon as possible (*MVP - Minimum Viable Product*).

Big Design Up-Front - requires the identification of all requirements and a complete architectural design before development begins (although architecture can evolve during development) which makes this strategy unwelcome in an agile approach. It can be considered in case of extreme risk, but is more often driven by the lack of agility of the user than by the technical risk.

Frameworks & templates - means the use of software frameworks, templates, and reference architectures, providing standard solutions for standard problems and reducing the number of architectural decisions. The use of the *Convention over Configuration* (CoC) paradigm reduces complexity since many architectural decisions are embedded in a framework, so formerly architectural decisions are now considered design decisions, and an easier change in architectural decisions is of great use in the application of agile methods. It should be aware that a framework does not always represent a comprehensive solution, so this strategy is combined with other strategies (Waterman *et al*., 2015).

Understanding the role, responsibilities and competencies of an architect

The traditional role of an architect is changing, which implied deep domain knowledge, a high level of abstraction in defining the structure, and an implicit or explicit right to make a decision in the field of interest (Poort, 2016). The architect focuses on delivering architecture as a service (Blair et al., 2010; Faber, 2010), coding according to needs, transferring technical knowledge (Babar, 2009), assisting the team in "*violation of rules*", providing communication on minimum documentation, with a division of responsibilities on the quality attributes (architect) and functionality (team) (Faber, 2010).

The architect must be a good communicator (Faber, 2010), while management skills are important in large distributed teams (Babar, 2009). The key role of the architect is to focus on the question "*What is blocking the agility of the team?*" (Buschmann & Henney, 2013) and to remove any obstacle that blocks the team's agility and frustrates the stakeholders (Mirakhorli & Cleland-Huang, 2013). He must have a clear architectural vision (Buschmann, 2012), to think beyond structure and technology by dealing with the structure of organization and production infrastructure (Nord et al., 2014), spending the most time with people living with his decisions (Buschmann, 2012).

The architect is involved in all phases of the development process while delaying his decisions (Hadar & Sherman, 2012). Coding as needed, giving preference to mentoring and teaching in relation to documentation, spends time with the user to understand how the system will be used, and the real requirements are derived from feedback from the users (Mirakhorli & Cleland-Huang, 2013). Although decisions can be made by the team, the architect takes care that the decisions

are consistent throughout the system, even when multiple teams work simultaneously, thus protecting the conceptual integrity of architecture and design. The architect has leadership qualities (Buschmann, 2012), technical knowledge and experience in design (Buschmann & Henney, 2013).

In the digital era, the role of the architect, which evolved from a specialist in a traditional architectural domain to a solution architect, was changed (Erder & Pureur, 2016; Zimmermann, 2016a). The architect must take into account scaling, complexity and distribution (Britto et al., 2016), and focus on system design that will be incrementally tested and quickly delivered in different environments (development, testing, production) across different platforms, private and public clouds (Erder & Pureur, 2016), where hardware infrastructure is added to the scope of competencies of the architect (Hohpe et al., 2016).

The architect becomes a connecting element, building bridges between teams and different levels of organization through leadership (Hohpe et al., 2016) and taking responsibility for design decisions that are risky, costly, and difficult to change (Woods, 2015), with constant management and monitoring of the current state of architecture (Holmes & Nicolaescu, 2017). The architect takes into account the complete life cycle of the software product, understands the source code, works in a decentralized manner, processes the government through the delivery process, removes all obstacles and provides resources for product delivery, directing and delivering timely decisions, minimizing multitasking and ensuring a coherent and sustainable product architecture (Erder & Pureur, 2016).

Leadership, mentoring and the translation of complex concepts into understandable concepts become more important than ever before (Hohpe et al., 2016). Different architectural competences are required at various management levels: technical skills, domain expertise, communicativeness and charisma (Martini & Bosch, 2016). The architect must have extensive knowledge of business domain and technology, as well as current architectural and agile practices (Holmes & Nicolaescu, 2017). He must be able to transfer and combine knowledge from isolated domains, must have broad views, including views on other industries as a source of new ideas (Hohpe et al., 2016).

The architect must act quickly and facilitate decision-making in an uncertain environment, in which knowledge and experience are needed, nontechnical skills such as communication and the ability to operate in ambiguous contexts are increasingly critical (Erder & Pureur, 2016). Due to the increasing "*need for speed*" in the digital era, additional skills are needed, as architects are involved in development, operations and maintenance, so they need to improve their business, financial, communication and educational skills.

In recent years, the trend of inversion of specialization is evident, where, for example, development of microservices or certain functionalities, requires full-stack developers that combine the skills of database design, integration, business logic of the domain and user interface, so that the role of the architect becomes virtual, i.e. becomes the responsibility of the team. Although increasing the complexity of the technology does not support this trend, time will show whether such a trend will be sustainable over the long term.

Generalists are necessary in distributed systems to deal with cross-cutting aspects such as: development of cross-domain software solutions, service deployment and testing (Pautasso et al., 2017a). At the same time, the development of the middleware platform reduces the need for architectural decisions, and by increasing the capacity and possibilities of a collaborative development environment, modern software tools further reduce the need for an architect (Hohpe et al., 2016).

Traditionalization of agile approach

In order to overcome the gap between speed and stability, traditional and agile methodologies are combined (Nord & Tomayko, 2006), in so-called traditionalization of the agile process of software development using architectural practices (Ambler, 2002; Babar, 2009; Erdogmus, 2009; Abrahamsson et al., 2010; Faber, 2010; Kruchten, 2010; Madison, 2010; Matković, Tumbas & Sakal, 2011) while at the same time combining various agile methods, techniques and practices (Ramakrishnan, 2010).

Agile approach is inclusive (Hayata et al., 2012), so ethno-relativism gradually prevails over ethno-centrism in the relation of agility and architecture (Kruchten, 2010), resulting in a wide application of architectural principles and practices in the agile process of development (Miyachi, 2011; Bellomo, Kruchten, et al., 2014; Eckstein, 2014; Nord et al., 2014), including risk

analysis (Ozkaya et al., 2013) and architectural modeling (Durdik, 2011; Prause & Durdik, 2012).

In the digital era, the application of architectural principles and practices in the agile development process, where ones are used in the conditions of the desired state, and the others outside the conditions of the desired state (Bellomo et al., 2013), is needed, but not sufficient. It is necessary to apply combined practices from other areas (from management to engineering), as well as the transformation of the development process according to the situational context (O'Connor et al., 2017).

Application of lean principles and practices

Lean principles enable efficient management in the agile development process by means of a management mechanism that motivates and supports IT professionals to do what an organization deems necessary (Ambler, 2009). Lean architecture emphasizes the importance of the form in relation to the structure. It focuses on the requirements of not only the users, but also the broader stakeholders, and along with the static, it includes a dynamic component, too (Booch, 2010). While the agile approach has a focus on speed, lean is focused on the "*right way*", avoiding premature optimization, timely thinking and planning through the early engagement of the team, domain experts and users in the architectural design (Hayata et al., 2012).

The combination of agile and lean practices, and the application of a lean concept for flow management, enables visualization, monitoring of technical debt and errors, balancing the allocation of critical architectural tasks and improving process flows, contributing to the reduction of total delay and number of errors/corrections (Buschmann, 2012). The application of lean approach has the potential to unify architecturally important tasks with functionalities, as opposed to agile methods that mainly create artificial boundaries, so defining tasks becomes a major problem (Nord et al., 2012).

Lean principles and thinking mean a focus on the complete product life cycle, i.e. combining design, development, delivery and validation in one feedback loop focusing on finding and delivering value, and continuous learning. By developing new business models (for example, *Two-sided market* & *Creating engaging experience*), the deeper understanding of what the user wants to do and how the software can help him with it (*design thinking*) is increasingly important, where the key to success is the ability of a gradual change (Poppendieck & Cusumano, 2012).

In the digital era, in order to achieve a continuous flow of values through the organization (*end-to-end*), lean thinking begins with an understanding of the flow of values and possible barriers in the course of value, with architecture being an integral part of this process. In this regard, it is important to use proactive (leading) metrics for the quality of current flows that can identify hidden obstacles in flow of value (Power & Conboy, 2015). Open, lean and sustainable architectural practices and techniques are required to build comprehensive and understandable frameworks, including sophisticated DevOps lean infrastructure (*service deployment pipeline*) with service monitoring, adapted for decentralized continuous delivery of value (Zimmermann, 2016a).

Combining agile practices and *lean start-up* that will support the flow of values from concept to production, and continuous learning from user experience, in a continuous development cycle (*build-measure-learn*), are crucial for the software product to go on the market and provide long-term survival (Hohpe et al., 2016; Pautasso et al., 2017a).

Application of continuous principles and practices

Madni (2008) emphasizes the continuous and incremental deployment, as the second most important principle of agile architecture, while for Isham (2008) continuous integration (CI) significantly reduces complexity and risk. In order to balance the agility and architecture, a continuous focus on architecture and continuous refactoring (CR) is necessary (Erdogmus, 2009; Booch, 2010), while the evolutionary and continuous development process and the exchange of ideas are crucial to achieving essential architecture (Blair et al., 2010).

Speed drives everything else, therefore continuous delivery (CD), with increased focus on CI (Bellomo, Ernst, et al., 2014; Nord et al., 2014), becomes important to enable continuous flow (CF) delivering new software to the production environment in a safe and reliable way (Poppendieck & Cusumano, 2012), even in the case of critical and very large online systems (Nord et al., 2014). Therefore, interest in the CD is growing steadily (Bellomo, Ernst, et al., 2014). On the other hand, Continuous Learning (CL) minimizes the effort to create functionality that

users will not need (YAGNI) (Poppendieck & Cusumano, 2012). CR is needed for good architecture (Fontdevila & Salias, 2013; Mirakhorli & Cleland-Huang, 2013), even sufficient for emergent architecture, if certain contextual factors are met (Chen & Babar, 2014).

In the digital era, CR continues to play a significant role (Hohpe et al., 2016; Holmes & Nicolaescu, 2017), CI tools are applied (O'Connor et al., 2017) with the continuous improvement of the agile development process (Scrum AND) integrating test-driven practices, such as automated *Test-Driven Development* (TDD) and CI, focusing on *built-in-product* quality attributes: modifiability, performance, availability, interoperability, security, usability, testability and deployability (Bellomo, Gorton & Kazman, 2015). The CD attracts great attention due to its potential (Bellomo et al., 2015; Chen, 2015; Erder & Pureur, 2016; Hohpe et al., 2016; Zimmermann, 2016b; O'Connor et al., 2017; Pautasso et al., 2017a, 2017b), such as: faster market entry, production of the "*right*" product, improved productivity and efficiency, better product quality, more satisfied users (Chen, 2015).

In this regard, the deployability feature appears as a completely new concept or quality attribute, which aims to reduce complexity and shorten the cycle, in the form of small, incremental, automated and reliable delivery (Bellomo et al., 2015) that enable a continuous flow of values (Power & Conboy, 2015). Architectural challenges arise due to CD (Chen, 2015), decentralized CD (Zimmermann, 2016b) and the appearance of microservices.

Microservices eliminate the so-called *single point of failure* using a CD strategy that only changes individual microservices while delivering it, without interruption, in others (Chen, 2015), allowing rapid scaling and delivery of applications for millions of users on cloud platforms (Villamizar et al., 2015). In a continuous family, a new practice, called Continuous Architecting (CA) emerged (Erder & Pureur, 2016; Martini & Bosch, 2016; Holmes & Nicolaescu, 2017), which presents a set of rules, architectural styles and tools that help the rapid delivery of software, supported by architectural principles (Holmes & Nicolaescu, 2017).

Use of architectural styles, design patterns and components

Applying component-based and pattern-based approaches is essential for building an intentional architecture (Leffingwell et al., 2008), while architectural style and principles should guide implementation, taking into account the form, not just the structure (Booch, 2010). Agile methods do not support the reuse of patterns, planning of reference architecture and components, or the development of a product line. Therefore, it is necessary to combine agile methods and architectural modeling with patterns and components (Durdik, 2011).

Architectural dynamics and agile principles can be supported by architectural patterns in order to avoid BDUF, e.g. *Sashimi pattern* or *Concentric approach* (Fontdevila & Salias, 2013). Architecture can be viewed in the light of the application of patterns and tactics that affect the time and cost of implementation, testing and delivery of changes. It is possible to apply different styles, patterns and tactics (*N-tier*, *client-server*, *SOA*, *publish-subscribe*...) to achieve agile architecture (Bellomo, Kruchten, et al., 2014), bearing in mind that strong components dependency is obstacles for continuous integration (Bellomo, Ernst, et al., 2014).

In the digital era, microservices and SOA, or (micro) service design patterns and principles, are an integral part of digital frameworks with quality stories, C4 architectural modeling, decision sharing (Y-statement), architecturally visible coding style, architectural refactoring and architectural roadmap (Zimmermann , 2016a). In modern Internet-based systems, which are flexible, dynamic, and based on microservices, long-term sustainable architecture is more a set of patterns and principles than a static, stable structure (Woods, 2016). Therefore, the need for the application of architectural styles (Pautasso et al., 2017b), design patterns and principles (Britto et al., 2016), at different levels, from architecture, through design to implementation (Gerdes et al., 2016), is greater than ever.

Decomposition and granularity

Attribute-driven design (ADD) method supports horizontal (*breadth-first*) and vertical (*deep-first*) approach to decomposition of design, which depends on the business context, domain knowledge and application technology (Nord & Tomayko, 2006). Although the feature/business-centric decomposition should be the primary

approach, for more efficient delivery of the project, the decomposition in line with architectural boundaries and frameworks should once again take priority, because iterations over architectural boundaries can open too many simultaneous challenges by increasing risk, such as simultaneous work with a large number of technologies. Therefore, the decomposition must be in line with the nature of the software product (Madison, 2010).

There are more approaches to the decomposition of domain problems and architectural aspects (Durdik, 2011). Unlike functionality, design is not easy to decompose into smaller tasks or "*technical stories*" and engage in agile practices (Bellomo, Kruchten, et al., 2014). The focus on horizontal decomposition (infrastructure and system elements, as well as common services) is needed in conditions of unstable infrastructure and production environment.

The more stable the infrastructure (platform, framework, tools) is, it is possible a better functional, vertical decomposition, which reduces the need for communication and coordination, and allows for better synchronization of the teams for parallel work (Nord et al., 2014). Expanded functionality should be divided into smaller increments that enable fast delivery of value to the user and fast feedback from users (Chen & Babar, 2014).

In the digital era, SOA and microservices, i.e. services of different granularity (macro & micro), reduce dependency with the help of vertical decomposition, allowing their independent development, testing and delivery (Villamizar et al., 2015; Poort, 2016; Zimmermann, 2016b; Holmes & Nicolaescu, 2017; Taibi et al., 2017), where the decomposition approach is chosen based on the context, vision, requirements and criteria of dependency, where DDD is one, but not the only one (Pautasso et al., 2017a).

Bearing in mind that the system will change over time for sure and that the service decomposition (or composition) is a reaction to the change of domain problems, the services should be designed to be updatable and/or rejectable, with the granularity evolving according to the requirements and experience, i.e. it should not be dictated by the choice of architectural style (Pautasso et al., 2017b). It should be kept in mind that granularity has an effect on performance (Pautasso et al., 2017a; Shadija et al., 2017).

An evolutionary approach

An evolutionary approach to the development of architecture is essential, whether seen as the evolution of intentional architecture (Ambler, 2002), the evolution of the minimal (walking skeleton) architecture (Abrahamsson et al., 2010), a continuous evolutionary process that results in an essential architecture (Blair et al., 2010), an evolutionary approach to the design of prototype and delivery (Nord & Tomayko, 2006) or a continuous incremental-iterative evolution that inhibits the erosion forces for the survival of every economically viable system (Booch, 2007; Erdogmus, 2009). The achievement of the evolutionary architecture of complex systems, as one of the principles of agile architecture (Madni, 2008), requires integration of architectural principles and agile approach (Babar, 2009).

The possibility of the evolution of the system is one of the advantages of software architecture (Durdik, 2011), which is important both for long-term systems and when the system should be translated from the state of "as-is" to the state of "to-be" (Ozkaya et al., 2013). Rare are systems that are built from scratch, but the existing architecture, which is becoming the subject of continuous evolution (Mirakhorli & Cleland-Huang, 2013), is mainly used, and an evolutionary approach is also suitable for building a core product line architecture (Harper & Dagnino, 2014).

For an evolutionary approach, those practices that enable small changes (increments) in short iterations, as well as techniques that allow fast feedback and learning, should be kept in mind, with the constant evolution of the requirements and their interrelations (Bellomo, Kruchten, et al., 2014; Nord et al., 2014), as well as the need for "*just enough anticipation*" (Poort, 2014).

In the digital era, software architecture evolves, from mainly technical discipline, to inclusion of business, sociological and cultural aspects, while, on the other hand, the rapid development of technology and new business models constantly moves the target in advance (Hohpe et al., 2016). ***The pressure to evolve software systems by delivering value at shorter time intervals is greater than ever*****,** instead of 1 or 2 times per year, a competitive market requires weekly, daily or even shorter delivery times (O'Connor, et al., 2017).

There is a growing consensus that good architectural foundations allow rapid, reliable and sustainable evolution of complex software using

an iterative-incremental approach (Bellomo et al., 2015; Woods, 2016). The application of the evolutionary design (Bellomo et al., 2013; Zimmermann, 2016b; Pautasso et al., 2017a) implies alignment with decisions that are later difficult to change and the application of tactic "*start stupid and evolve*" (Pautasso et al., 2017b), while avoiding architectural changes in each individual iteration, as this leads to cost increases (Waterman et al., 2015), but defects need to be timely identified in order to preserve the long-term evolution and sustainability of architecture (Britto et al., 2016).

Evolution of components (Woods, 2015) and services (Zimmermann, 2016b) implies the application of the principle of backward compatibility, so that the changes affect users less. Evolution and change should get the lead, with timely anticipation of future architecturally significant events and their evolution point of view (Poort, 2016). It seems that the next phase, the so-called dynamic evolution, be even more radical, with intelligent dynamic compositions, cloud platforms, and linking IoT (Woods, 2016).

Quality attributes - continuous focus and prioritization

Traditional architectural methods (QAW, ADD, ATAM, CBAM) put an early focus on quality attributes (Nord & Tomayko, 2006), as opposed to agile methods that focus on fast delivery of value and enable improvement in so-called "*3 big*": quality, productivity and morale (Leffingwell et al., 2008). While the traditional approach is used to evaluate the quality attributes, advocates of the agile approach argue that refactoring helps to acquire the quality attributes. Quality attributes and their prioritization are not in the focus of agile approach, as they are often not a measure of success, but a focus on functionality, budget and delivery deadlines (Babar, 2009).

Quality attributes should be in the focus as soon as possible, with the division of responsibilities to the architect (quality attributes) and development team (functionality) (Faber, 2010). A combination of architectural and agile techniques is needed to achieve a balance between business (functionality) and architectural (quality attributes) priorities (Madison, 2010). Undefined quality attributes are the causes of design, documentation and code problems (Prause & Durdik, 2012). In order to support a continuous flow of values, prioritization needs to consider the dependence between functional stories and non-functional requirements (quality attributes), and consequently, the dependent functional requirements earlier in development should be withdrawn (Buschmann, 2012; Nord et al., 2012; Fontdevila & Salias, 2013).

Quality attributes are an integral part of risk analysis and architecture evaluation (e.g. *QA utility tree*) (Ozkaya et al., 2013). Identifying and prioritizing quality attributes, as a continuous process, is crucial for the implementation of valuable functionalities without the risk of intensive re-design or complex coordination between multiple teams (Nord et al., 2014). The possibility of continuous and rapid delivery (deployability) can also be considered as a quality attribute (Bellomo, Ernst, et al., 2014).

In the digital era, and the Internet-connected system, the focus is on the quality attributes, agility and decision-making (Hohpe et al., 2016; Woods, 2016; Holmes & Nicolaescu, 2017), so architectural requirements are involved in sprint planning and demo prototyping, in such a way that explicit attention to architecture allows long-term modifiability and evolution (Britto et al., 2016). Lack of prioritization of quality attributes leads to problems in security, monitoring, and integration (Woods, 2015). In the CD context, the priorities of individual quality attributes are increased: deployability, security, loggability, modifiability, monitorability, testability (Chen, 2015), as they ensure that architecture is optimized for different phases of the development cycle (Holmes & Nicolaescu, 2017).

For the users, the most important are the following quality attributes: functionality, quality, availability, ease of use, performance, variability, safety, interoperability and simple testing (Tumyrkin, Mazzara, Kassab, Succi & Lee 2016), where prioritization of quality attributes positively affects the reduction of documentation (Gerdes et al., 2016).

Microservices are a good choice if they meet the required quality assignments, but account must be taken of possible changes in the requirements, especially when it comes to security, as well as a balance of flexibility and complexity (Holmes & Nicolaescu, 2017).

Application of tools and technologies in the combined ecosystem

Although Ambler (2002) puts focus on people, communication and techniques, a good choice of implementation technology can simplify development, improve system extensibility and

ease of use (Leffingwell et al., 2008). Traditionally, there was a strict division of applications and support infrastructure (testing, configuration and management tools, deployment scripts, and other components), which was not considered as an integral part of the system.

However, by applying combined practices, with the support of modern tools, this division between the development and production environment is increasingly being eradicated by forming a combined ecosystem (Bellomo, Ernst, et al., 2014) where CD is primarily used instead of a project approach, and architecture must be designed from the beginning to support dynamic updates and continuous changes (Poppendieck & Cusumano, 2012).

Fontdevila and Salías (2013) emphasize the importance of technology and tools at four levels: frameworks and testing tools, quality assurance tools, tools for monitoring metrics about flexibility and long-term maintenance (*maintainability*), and deployment tools & configuration, while Mirakhorli and Cleland-Huang (2013) suggest the use of code and design testing tools. Application of the DevOps concept involves merging development and operations into one team, using CD-enabled tools even in the case of large, online, critical systems (Nord et al., 2014).

In the digital era and the Internet-connected system, system of systems or software ecosystems are formed in all industries (Hohpe et al., 2016) with interconnections between commercial and custom-made software, hardware platforms and organizational entities, of which each has its own evolutionary cycle (Poort, 2016).

In such an environment, architectural decisions are also technology-related decisions about frameworks, language, platforms, etc. (Gerdes et al., 2016), whereby information is to be shared with simple tools (Woods, 2015). The architect must prove the benefit of new technologies by creating an executive prototype, as part of an architectural runaway (Erder & Pureur, 2016). Modern open source development tools, code management, testing, deployment, production, monitoring and configuration are gaining importance, while the contemporary effective software development process is hardly conceivable to the previous generation of developers (O'Connor et al., 2017).

For this reason, the focus of the researchers and practitioners has shifted from people and processes to integration technologies and platforms (*RESTFull API, cloud computing, DevOps*) (Zimmermann, 2016b). Observed from a modern architectural perspective, along with a set of architectural rules, it is also necessary to provide support tools, which will support both incremental and agile delivery methods such as CD (Holmes & Nicolaescu, 2017).

As the capabilities of development platforms and environments increase, development teams are increasingly accepting tools and practices that allow them to avoid major BDUF decisions, to divide them and eliminate dependencies, in which they help the intensive development of middleware platforms that reduce the need for architectural decisions by integrating them into technological environment, further reducing the need for an architect.

This trend raises the question of whether the need for an architect is lost, and some Internet-scaling companies (like Google and Spotify) have almost no positions with the name of the architect, with their architecture living in code, with documented decisions, managed through a version control system or code review tools that support visualization techniques and tools (Hohpe et al., 2016).

Microservices require automated deployment tools and the DevOps strategy (develop, test, deploy, operate, monitor) (Villamizar et al., 2015), so, the modern technological environment

Table 5 Agile architecture practices in the digital era

Practice	2001-2010	2011-2014	2015-2017
TDD / Testing / Automated testing	S06 S04 S08 S11	S23 S24 S27 S25 S28 S29 S34 S38	S39 S42 S43 S44 S45 S49 S50 S53 S59
Prototype / Experimentation (spikes)	S04 S11 S13	S23 S30 S31 S33 S35 S38	S39 S42 S44 S52 S58 S59
Lean thinking	S08 S14	S22 S23 S27 S31 S36	S41 S44 S51 S52 S54 S56
Incremental value delivery	S07 S11 S14	S23 S27 S34 S35 S36	S39 S40 S42 S43 S54 S57
Refactoring / Continuous Refactoring	S02 S03 S06 S07 S09 S11 S12 S14 S18	S19 S20 S23 S25 S30 S35 S36 S37	S53 S46 S56 S57 S58

Practice	2001-2010	2011-2014	2015-2017
Continuous Architecting / Zipper model	S09 S12 S16	S21 S30 S36 S38	S50 S53 S56 S57 S60
Revision of source code / Code & Sprint review	S06 S08 S14	S19 S24	S47 S49 S52 S53 S56
Evaluation of architecture and design	S08 S14 S15 S16 S17	S19 S21 S24 S25 S29 S30	S39 S44 S52 S56
Multi-level teams / Scrum of Scrums / SAFe / CAFFEA	S06	S28 S29 S35	S41 S49 S53 S57
Continuous Integration	S06	S27 S29 S37 S38	S39 S49 S52 S58
Continuous Learning	S13	S27 S31 S34	S49 S56 S58
Minimum Viable Architecture (MVA) / Walking skeleton	S12 S13 S14	S20 S25 S35 S38	S42 S44 S57
Organizing a team according to Convey's Law	S04	S31 S35	S54 S58 S59
Architectural Runaway	S04	S35 S36	S50 S51 S53
Architectural Roadmap	S08	S25	S46 S51 S56
Deffering architectural and design decisions	S15	S20 S24 S31 S33	S42 S44
Minimum documentation / Realized architecture documented in source code	S13 S18	S20 S30	S47 S50
Pair-programming	S06 S11	S19 S21 S25	S39
Common semantics / Metaphor / Common Language	S08	S35	S58
DevOps		S35	S45 S49 S50 S51 S52 S54 S56 S58 S59 S60
Continuous Delivery		S27 S35 S38	S40 S43 S45 S49 S50 S52 S56 S59 S60
Cloud Computing		S38	S45 S50 S47 S54 S55 S56 S61
Minimum Viable Product (MVP) / Lean Start-Up (build-measure-learn)		S20 S27 S30 S35	S42 S44 S50 S56 S58 S60
Small, autonomous and dedicated teams for work in a bounded context / 'Two-pizza' teams		S35	S39 S57 S58
Continuous Flow of Value (end-to-end)		S27	S41 S56 S59
Resolving interdependences		S23 S36	S39 S42 S51
Combined ecosystem / Dev, Build, Test, Oper, Prod tools & environments support		S27 S34 S35 S38	S45 S60
Design principles (SOLID, KISS, DRY / IDEAL)		S30 S37	S42 S44
RCDD (Risk & Cost Driven Design)		S36	S46 S51
System monitoring		S38	S53 S56
Kanban visualization / WIP limit		S23 S27	S41
Use of Microservices / SOA			S45 S46 S51 S52 S55 S56 S60 S61
Cloud-based services (IaaS, PaaS, SaaS / FaaS, Serverless)			S45 S55 S56 S59 S61
RESTFull API / HTTP			S45 S54 S56 S58 S61
Containerization			S45 S52 S54
API Gateways			S45 S54 S59
DDD (Domain Driven Design) / Bounded Context			S54 S56 S58
Independently deliverable (micro)services			S51 S54 S60
Monitorning of (micro)services			S45 S54 S60
Verzioning of (micro)services			S58 S59 S60

Source: Authors

favors the development of microservices, since each service can be designed, developed and shipped by a different team and on a different technological stack. In addition, the team is in charge of the complete development process of the service including deployment, operations and update.

However, the spread of microservices also increases the complexity of ecosystems (Taibi et al., 2017), so more work is needed on tool development and system-level design. In this regard, *serverless computing* is a new trend that aims to provide infrastructure for deployment and scaling services that are hidden from the developer (Pautasso et al., 2017b).

3.1. Agile architecture practices in the digital era

The results of the analysis of the selected publications are presented in order to get the answer is the question of RQ2: *What are the practices of developing and implementing Agile architecture in a modern digital environment?*

A total set of 40 practices have been identified, which are associated with at least three different sources (Table 5).

It can be noted that in the digital era there are many practices of agile architecture that are available for developing complex software systems in a modern development and production environment, and whose application depends on the particular context.

4. Discussion

In this section, the general considerations will first be discussed, and then the answers to the research questions. After that, the conducted research will be compared with other similar researches, the identified contradictory attitudes will be considered, as well as the possibilities for further research and the limitations of the conducted research.

General considerations: Out of 61 selected primary studies, 18 (30%) is from the period 2001-2010, 20 (33%) from the period 2011-2014 and 23 (37%) from the period 2015-2017, which is designated as a digital era. Out of the total, 32 (52%) studies were published in the magazine, 23 (38%) conference, 5 (8%) workshop and 1 (2%) as a whitepaper. The average grade of the selected primary studies is 8.4 on a scale of 10. Empirical studies are the most numerous and make up 22 (36%), followed by studies based on the experts' experience 21 (34%) and finally the experts' opinion 18 (30%).

Answers to research questions: The *10 key challenges* of agile architecture in the digital era have been identified, where the most important new challenge for the digital era is the application of microservice architecture that follows the challenges associated with new business models of the digital economy. Very significant challenges from the previous periods are: balancing agility and architecture, organization, communication and coordination, as well as the challenges of scaling. In addition, the following challenges are identified as important: interdependence of components, documentation of architecture, preservation of conceptual integrity and consistency of architecture, interdependence of requirements, and product lifecycle management (process optimization and value stream).

There should be noted that the old challenges remain with the emergence of new ones. In the coming period, a growing trend can be expected for the challenges brought about by the interdependence of components, due to the ever-increasing complexity in the conditions of the scaled, distributed environment, as well as the challenges that will bring new business models of the digital economy into future intelligent-connected systems (Woods, 2016).

The *10 key success factors* of agile architecture in the digital era have been identified, of which the most important are: application of continuous practices, evolutionary approach, application of tools and technologies in a combined ecosystem, continuous focus and quality attributes prioritization, understanding of the role, responsibilities and competencies of the architect. Following success factors are identified as important, too: decomposition and granularity, understanding of the context and choice of the implementation strategy, use of architectural styles, design patterns and components, application of lean principles and practices, and traditionalization of the agile approach.

The growing trend of almost all of the identified success factors, especially the importance of tools and technologies in the combined ecosystem, as well as the application of continuous practices, including continuous architecting, is also evident. That can be interpreted as achieving a certain level of maturity of the agile approach traditionalization process, which shows a downward trend.

The *40 practices* of agile architecture in the digital era have been identified, which have been referenced in at least three primary studies.

The most relevant current practices identified in the period (2001 - 2017) are: TDD/testing/ automated testing, prototype/experimentation (spikes), Lean thinking and incremental delivery.

The most significant new practices identified

in the period (2011 - 2017) are: DevOps, Continuous Delivery, Cloud computing and the Minimum Viable Product (MVP)/Lean Start-Up.

The most significant new practices identified in the period (2015-2017) are: Use of microservices/SOA, Cloud-based services (*IaaS, PaaS, SaaS/FaaS, Serverless*), RESTFull API/HTTP, Containerization, API Gateways, DDD (Domain Driven Design)/Bounded Context, independently delivered (micro) services, monitoring (micro) services and versioning (micro) services. It is interesting that two studies identify the need for machine learning, i.e. artificial intelligence (AI) because AI increasingly affects the labor market (Vochozka, Kliestik, Kliestikova & Sion, 2018) and contributes to the expansion of digitally mediated labor in a platform-based economy (Mitea, 2018).

Such a large number of identified challenges, success factors and practices indicate that there is no one solution applicable to each problem. It is necessary to transform, combine and balance different approaches, methods, principles, practices, tools, and technologies in order to give better answers to the challenges that bring a specific context.

Comparison with other similar research: According to available information, there is no SLR that deals with the trends and practices of agile architecture in the digital era. There is a SLR that deals with a similar or related topic (Dikert, Paasivaara & Lassenius, 2016), which explores the challenges and success factors of organizations' transformation in the adoption of an agile and lean process of software development in scaling conditions, citing 35 challenges in 9 categories, and 29 success factors, of which the most important is management support, choosing and customizing the agile model, training and coaching, mindset and alignment.

Also, there is a Systematic Mapping Study (Yang, Liang & Avgeriou, 2016) that combines architectural and agile methods including architectural activities and approaches, agile methods and practices, cost, benefits, challenges, success factors, tools, and lessons learned. The main difference is that the above research focuses on the software development phase, while the realized SLR in this paper focuses on all stages of the life cycle of the product: development, operation and production, i.e. a complete development and production environment, bearing in mind that the agile architecture, by the emergence of CD, has passed the boundaries of the software development phase.

Opposite Attitudes: The development of technology and tools is the main reason that a modern hyper-agile, lean development process in the combined ecosystem is possible at all; therefore, O'Connor et al. (2017) raises the question of whether this observation is in opposition to the Agile Manifesto, i.e. the principle of "*Individuals and interaction, before processes and tools*".

Babar (2009) notes that the responsibility, regarding the specification of the requests, is shifted to the user. However, Mirakhorli & Cleland-Huang (2013) claim that the time has elapsed when the specification of the requests was obtained from users, but it is necessary to be with the user, understand how the system will be used, put the minimum viable product into the user's hands, and evolve into short iterations with continuous measurement and learning.

Opportunities for further research: The identified primary studies have been largely based on the opinions and experience of the experts. Therefore, for future research, case studies of successful and unsuccessful implementation of agile architecture in the digital era are proposed. The development of large distributed software systems is significantly different from the development of small systems, whereby experience in the development of large systems is difficult to obtain, while learning programming on the example of small systems can be problematic. Therefore, future research could study in more detail the problems of acquiring competencies for successful implementation of agile architecture in complex distributed systems.

There is a lack of methods and guidelines for the implementation of agile architecture using microservices in the development of complex distributed systems, so future research could address this problem. The design of complex distributed software systems is inherently demanding, so for the successful implementation of agile architecture using microservices, SOA principles and practices must be combined with modern software development practices (Zimmermann, 2016b), so it is necessary to explore how to combine microservices and SOA principles and practices in the context of hyper-agile, lean development of complex software.

Research Limitations: In order to reduce the risk of bias, more researchers are involved in the development and evaluation of research protocols,

including the inclusion/exclusion criteria, database queries and the selection of primary studies. In order to reduce the risk of the subjectivity of researchers in the selection (coding), analysis and synthesis of data, the Atlas.ti software tool is used, more researchers are involved in evaluating the results of the research, while traceability is enabled by reference to any claims stated in relation to the results of the research.

In order to reduce the risk of omitting essential studies due to limitations regarding database searches (selection of logical operators and query keywords), the query is specifically adapted to each database. In addition, snowballing was applied, based on which relevant studies were added. One should also bear in mind the risk of the bias of the authors of the primary studies since successful examples of the implementation of agile architecture prevail.

Conclusions

The paper presents the results of a systematic literature review related to the trends of challenges and success factors, as well as the practice of agile architecture in the digital era.

The 61 primary studies were selected and analyzed in the period from 2001 to 2017, which can be divided into three characteristic intervals: Large-scale agile period (2001-2010), Continuous Delivery/DevOps period (2011-2014) and Microservices period (2015- 2017).

The key **challenges** of agile architecture in the digital era are: the application of microservices, balancing agility and architecture, the structuring of organization, communication and coordination in order to support context scaling, minimum documentation and dealing with interdependency of requirements and components, while preserving the conceptual integrity and consistency of architecture in conditions of development of new business models of the digital economy and permanent changes.

The digital era, along with the old challenges of software development, including challenges in the development of distributed systems, is characterized by new challenges that arise with the emergence and application of microservices, both in terms of migration of monolithic applications into microservices using agile approach, as well as the development of greenfield software solutions in a distributed and scaled environment.

In response to challenges, the key **success factors** of agile architecture in the digital era are: understanding the context, choosing a strategy of implementation (or combining them) and choosing appropriate tools and technologies in a combined ecosystem, in order to support the application of an evolutionary approach, continuous and lean principles and practices, focusing on quality attributes and their prioritization. In this sense, understanding the role, responsibilities and competencies of the architect is crucial, regardless of whether this role is virtual (i.e. the responsibility of the team), whereby the architect should be given a key contribution in choosing the appropriate architectural style, the design patterns and components, the decomposition of the system to the optimal level of granularity and the application of the necessary architectural practices.

The 40 old and new **practices** of agile architecture in the digital era have been identified, which should be combined depending on the specific context in order to successfully overcome the challenges of agile architecture. Two studies also indicate the need for applying artificial intelligence practices, specifically machine learning.

It is an interesting finding that success factors in the period 2015 - 2017 are essentially the same as in previous periods, with the different significance and influence of individual factors, while new challenges of the digital era are followed by new practices.

A large number of challenges, success factors and available practices indicate that there is neither solution applicable to each problem nor the same solution can be repeated for the same problem in a different context.

In the digital era, agile architecture will mark microservices based on hyper-agile and lean approach, combined with SOA principles and practices, while the development of AI will bring new challenges and practices related to future, intelligently connected systems.

Future research should focus on agile architecture in the development of complex distributed systems, whereby more case studies with successful and unsuccessful implementation examples would be desirable. Research on the problem of acquiring competencies for the successful implementation of agile architecture in complex distributed systems is needed, followed by research that will propose methods and

guidelines for the implementation of agile architecture using microservices, as well as research on the possibilities of combining microservices and SOA principles and practices in the context of hyper-agile, lean development of complex software, especially in the development of complex distributed software systems.

Acknowledgements

This work was supported in part by the Provincial Secretariat for Higher Education and Scientific Research under the Grants 142-451-2477/2018-02.SM

References

Abrahamsson, P., Babar, M. A. & Kruchten, P. (2010). Agility and Architecture: Can They Coexists? *IEEE Software, 27(2),* 16-22. https://doi.org/10.1109/MS.2010.36

Ambler, S. W. (2002). Lessons in agility from internet-based development. *IEEE Software*, *19(2),* 66-73. https://doi.org/10.1109/52.991334

Ambler, S. W. (2009). Scaling agile software development through lean governance. In *Proceedings of the 2009 ICSE Workshop on Software Development Governance, SDG 2009* (pp. 1-2). IEEE. https://doi.org/10.1109/SDG.2009.5071328

Babar, M. A. (2009). An exploratory study of architectural practices and challenges in using agile software development approaches. In *2009 Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture* (pp. 81-90). IEEE. https://doi.org/10.1109/WICSA.2009.5290794

Bellomo, S., Ernst, N., Nord, R. & Kazman, R. (2014). Toward design decisions to enable deployability: Empirical study of three projects reaching for the continuous delivery holy grail. In *Proceedings - 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014* (pp. 702-707). IEEE. https://doi.org/10.1109/DSN.2014.104

Bellomo, S., Gorton, I. & Kazman, R. (2015). Towards Agile Architecture: Insights from 15 Years of ATAM Data. *IEEE Software, 32(5)*, 38-45. https://doi.org/10.1109/MS.2015.35

Bellomo, S., Kruchten, P., Nord, R. L. & Ozkaya, I. (2014). How to agilely architect an agile architecture. *Cutter IT Journal of Information Technology Management*, 27(2), 12-17.

Bellomo, S., Nord, R. L. & Ozkaya, I. (2013). A study of enabling factors for rapid fielding combined practices to balance speed and stability. In *35th International Conference on Software Engineering (ICSE)* (pp. 982-991). IEEE Press. https://doi.org/10.21236/ADA591481

Blair, S., Watt, R. & Cull, T. (2010). Responsibility-driven architecture. *IEEE Software, 27(2)*, 26-32. https://doi.org/10.1109/MS.2010.52

Booch, G. (2007). The Economics of Architecture-First. *IEEE Software*, *24*(5), 18-20. https://doi.org/10.1109/MS.2007.146

Booch, G. (2010). An architectural oxymoron. *IEEE Software. 27(5)*, 96-96. https://doi.org/10.1109/MS.2010.117

Britto, R., Šmite, D. & Damm, L. O. (2016). Software Architects in Large-Scale Distributed Projects: An Ericsson Case Study. *IEEE Software, 33(6)*, 48-55. https://doi.org/10.1109/MS.2016.146

Buschmann, F. (2012). A Week in the Life of an Architect. *IEEE Software, 29(3)*, 94-96. https://doi.org/10.1109/MS.2012.55

Buschmann, F. & Henney, K. (2013). Architecture and agility: Married, divorced, or just good friends? *IEEE Software, 30(2)*, 80-82. https://doi.org/10.1109/MS.2013.25

Chen, L. (2015). Towards Architecting for Continuous Delivery. In *Proceedings - 12th Working IEEE/IFIP Conference on Software Architecture* (pp. 131-134). IEEE. https://doi.org/10.1109/WICSA.2015.23

Chen, L. & Babar, M. A. (2014). Towards an evidence-based understanding of emergence of architecture through continuous refactoring in agile software development. In *2014 IEEE/IFIP Conference on Software Architecture* (pp. 195-204). IEEE. https://doi.org/10.1109/WICSA.2014.45

Coplien, J. & Bjørnvig, G. (2010). *Lean Architecture for Agile Software Development*. Cornwall, UK: John Wiley & Sons.

Dikert, K., Paasivaara, M. & Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations: A systematic literature review. *Journal of Systems and Software, 119*, 87-108. https://doi.org/10.1016/j.jss.2016.06.013

Durdik, Z. (2011). Towards a process for architectural modelling in agile software development. In *Proceedings of the joint ACM SIGSOFT conference--QoSA and ACM SIGSOFT symposium--ISARCS on Quality of software architectures--QoSA and architecting critical systems--ISARCS* (pp. 183-192). ACM. https://doi.org/10.1145/2000259.2000291

Dybå, T. & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and software technology, 50(9-10),* 833-859. https://doi.org/10.1016/j.infsof.2008.01.006

Eckstein, J. (2014). Architecture in large scale agile development. In *International Conference on Agile Software Development* (pp. 21-29). Springer, Cham. https://doi.org/10.1007/978-3-319-14358-3_3

Erder, M. & Pureur, P. (2015). *Continuous Architecture: Sustainable Architecture in an Agile and Cloud-Centric World*. Burlington, US: Morgan Kaufman.

Erder, M. & Pureur, P. (2016). What's the Architect's Role in an Agile, Cloud-Centric World? *IEEE Software, 33(5)*, 30-33. https://doi.org/10.1109/MS.2016.119

Erdogmus, H. (2009). Architecture meets agility. *IEEE Software, 26(5)*, 2-4. https://doi.org/10.1109/MS.2009.121

Faber, R. (2010). Architects as service providers. *IEEE Software, 27(2)*, 33-40. https://doi.org/10.1109/MS.2010.37

Falessi, D., Cantone, G., Sarcia, S. A., Calavaro, G., Subiaco, P., & D'Amore, C. (2010). Peaceful coexistence: Agile developer perspectives on software architecture. *IEEE Software, 27(2)*, 23-25. https://doi.org/10.1109/MS.2010.49

Fontdevila, D. & Salias, M. (2013). Software Architecture in the Agile Life Cycle. *The Advances in Computer Science : an International Journal (ACSIJ), 2(1)*, 48-52.

Ford, N., Parsons, R. & Kua, P. (2017). *Building Evolutionary Architectures - Support Constant Change*, Sebastopol, US: O'Reilly Media, Inc.

Fowler, M. & Lewis, J. (2014). Microservices. Retrieved January 17, 2018 from: https://martinfowler.com/articles/microservices.html

Gartner (2014). Gartner Says in the Digital World CIOs Need Bimodal IT: Rock Solid IT with Ability for Fluidity. Retrieved January 17, 2018 from: https://www.gartner.com/en/newsroom/press-releases/2014-10-06-gartner-says-in-the-digital-world-cios-need-bimodal-it-rock-solid-it-with-ability-for-fluidity

Gerdes, S., Jasser, S., Riebisch, M., Schröder, S., Soliman, M., & Stehle, T. (2016). Towards the essentials of architecture documentation for avoiding architecture erosion. In *Proccedings of the 10th European Conference on Software Architecture Workshops* (p. 8). ACM. https://doi.org/10.1145/2993412.3004844

Hadar, I. & Sherman, S. (2012). Agile vs. plan-driven perceptions of software architecture. In *2012 5th International Workshop on Co-operative and Human Aspects of Software Engineering (CHASE)* (pp. 50-55). IEEE. https://doi.org/10.1109/CHASE.2012.6223022

Harper, K. E. & Dagnino, A. (2014). Agile Software Architecture in Advanced Data Analytics. In *2014 IEEE/IFIP Conference on Software Architecture* (pp. 243-246). IEEE. https://doi.org/10.1109/WICSA.2014.16

Hayata, T., Han, J. & Beheshti, M. (2012). Facilitating agile software development with lean architecture in the DCI paradigm. In *2012 Ninth International Conference on Information Technology-New Generations* (pp. 343-348). IEEE. https://doi.org/10.1109/ITNG.2012.157

Hohpe, G., Ozkaya, I., Zdun, U. & Zimmermann, O. (2016). The Software Architect's Role in the Digital Age. *IEEE Software*, 33(6), 30-39. https://doi.org/10.1109/MS.2016.137

Holmes, B. & Nicolaescu, A. (2017). Continuous Architecting: Just another buzzword? *Full-scale Software Engineering/The Art of Software Testing*, 1-6.

Isham, M. (2008). Agile architecture IS possible - You first have to believe! In *Agile 2008 Conference* (pp. 484-489). IEEE. https://doi.org/10.1109/Agile.2008.16

Jalali, S., & Wohlin, C. (2012). Systematic literature studies: Database searches vs. backward snowballing. In *Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 29-38). IEEE. https://doi.org/10.1145/2372251.2372257

Kitchenham, B. (2004). Procedures for Performing Systematic Reviews. *Keele, UK, Keele University, 33(2004),* 1-26.

Kitchenham, B. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. *EBSE Technical Report EBSE-2007-01, Keele University.*

Kruchten, P. (2010). Software Architecture and Agile Software Development—A Clash of Two Cultures? In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2* (pp. 497-498). ACM. https://doi.org/10.1145/1810295.1810448

Kruchten, P. (2013). Agile architecture. Retrieved January 17, 2018 from: https://philippe.kruchten.com/2013/12/11/agile-architecture

Leffingwell, D., Martens, R. & Zamora, M. (2008). *Principles of Agile Architecture*. Leffingwell, LLC. & Rally Software Development Corp.

Madison, J. (2010). Agile-Architecture Interactions. *IEEE Software*, 41-48. https://doi.org/10.1109/MS.2010.35

Madni, A. M. (2008). Agile Systems Architecting (ASA): Placing Agility Where it Counts. *Conference on Systems Engineering Research (CSER)* (pp. 1-7).

Martini, A. & Bosch, J. (2016). A multiple case study of continuous architecting in large agile companies: current gaps and the CAFFEA framework. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*(pp. 1-10). IEEE. https://doi.org/10.1109/WICSA.2016.31

Matković, P., Tumbas, P. & Sakal, M. (2011). The RSX model: traditionalisation of agility. *Strategic Management*, *16(2)*, 74-83.

Mirakhorli, M. & Cleland-Huang, J. (2013). Traversing the twin peaks. *IEEE Software, 30(2)*, 30-36. https://doi.org/10.1109/MS.2013.40

Mitea, D. R. E. (2018). The Expansion of Digitally Mediated Labor: Platform-Based Economy, Technology-Driven Shifts in Employment, and the Novel Modes of Service Work. *Journal of Self-Governance and Management Economics 6(4),* 7-13.

Miyachi, C. (2011). Agile software architecture. *ACM SIGSOFT Software Engineering Notes, 36(2)*, 1-3. https://doi.org/10.1145/1943371.1943388

Moore, E. & Spens, J. (2008). Scaling agile: Finding your agile tribe. *Agile 2008 Conference* (pp. 121-124). IEEE. https://doi.org/10.1109/Agile.2008.43

Newman, S. (2015). *Building Microservices*. O'Reilly Media, Inc.

Nord, R. L., Ozkaya, I. & Kruchten, P. (2014). Agile in Distress: Architecture to the Rescue. In *International Conference on Agile Software Development* (pp. 43-57). Springer, Cham. https://doi.org/10.1007/978-3-319-14358-3_5

Nord, R. L., Ozkaya, I. & Sangwan, R. S. (2012). Making architecture visible to improve flow management in lean software development. *IEEE Software, 29(5)*, 33-39. https://doi.org/10.1109/MS.2012.109

Nord, R. L. & Tomayko, J. E. (2006). Software architecture-centric methods and agile development. *IEEE Software*, *23(2)*, 47-53. https://doi.org/10.1109/MS.2006.54

O'Connor, R. V., Elger, P. & Clarke, P. M. (2017). Continuous software engineering—A microservices architecture perspective. *Journal of Software: Evolution and Process*, 29(11), 1-12. https://doi.org/10.1002/smr.1866

Ozkaya, I., Gagliardi, M. & Nord, R. L. (2013). Architecting for large scale agile software development: A risk-driven approach. *CrossTalk*, *26(3)*, 17-22.

Pautasso, C., Zimmermann, O., Amundsen, M., Lewis, J. & Josuttis, N. (2017). Microservices in Practice, Part 1: Reality Check and Service Design. *IEEE Software*, *34(1)*, 91-98. https://doi.org/10.1109/MS.2017.24

Pautasso, C., Zimmermann, O., Amundsen, M., Lewis, J. & Josuttis, N. (2017). Microservices in Practice, Part 2: Service Integration and Sustainability. *IEEE Software*, *34(2)*, 97-104. https://doi.org/10.1109/MS.2017.56

Poort, E. (2014). Driving agile architecting with cost and risk. *IEEE Software, 31(5)*, 20-23. https://doi.org/10.1109/MS.2014.111

Poort, E. (2016). Just Enough Anticipation: Architect Your Time Dimension. *IEEE Software, 33(6)*, 11-15. https://doi.org/10.1109/MS.2016.134

Poppendieck, M. & Cusumano, M. A. (2012). Lean software development: A tutorial. *IEEE Software, 29(5)*, 26-32. https://doi.org/10.1109/MS.2012.107

Power, K. & Conboy, K. (2015). A Metric-Based Approach to Managing Architecture-Related Impediments in Product Development Flow: An Industry Case Study from Cisco. In *Proceedings of the second international workshop on software architecture and metrics* (pp. 15-21). IEEE Press. https://doi.org/10.1109/SAM.2015.10

Prause, C. R. & Durdik, Z. (2012). Architectural design and documentation: Waste in agile development? In *2012 International Conference on Software and System Process (ICSSP)* (pp. 130-134). IEEE. https://doi.org/10.1109/ICSSP.2012.6225956

Ramakrishnan, S. (2010). On Integrating Architecture Design into Engineering Agile Software Systems Agile Methods in Practice - Combination of Techniques within an Agile Method. *Issues in Informing Science and Information Technology, 7*, 9-25. https://doi.org/10.28945/1229

Richardson, C. & Smith, F. (2016). Microservices: From Design to Deployment. Retrieved January 17, 2018 from: https://www.nginx.com/blog/microservices-from-design-to-deployment-ebook-nginx/

Shadija, D., Rezai, M. & Hill, R. (2017). Microservices: Granularity vs. Performance. In *Companion Proceedings of the10th International Conference on Utility and Cloud Computing* (pp. 215-220). ACM. https://doi.org/10.1145/3147234.3148093

Taibi, D., Lenarduzzi, V., Pahl, C. & Janes, A. (2017). Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In *Proceedings of the XP2017 Scientific Workshops* (p. 23). ACM. https://doi.org/10.1145/3120459.3120483

Tumyrkin, R., Mazzara, M., Kassab, M., Succi, G. & Lee, J. Y. (2016). Quality attributes in practice: Contemporary data. In *Agent and Multi-Agent Systems: Technology and Applications* (pp. 281-290). Springer, Cham. https://doi.org/10.1007/978-3-319-39883-9_23

Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S (2015). Evaluating the Monolithic and the Microservice Architecture Pattern to Deploy Web Applications in the Cloud. In *2015 10th Computing Colombian Conference (10CCC)* (pp. 583-590). IEEE. https://doi.org/10.1109/ColumbianCC.2015.7333476

Vochozka, M., Kliestik, T., Kliestikova, J. & Sion, G. (2018). Participating in a Highly Automated Society: How Artificial Intelligence Disrupts the Job Market. *Economics, Management, and Financial Markets 13(4),* 57-62.

Waterman, M., Noble, J. & Allan, G. (2012). How much architecture? Reducing the up-front effort. In *2012 Agile India* (pp. 56-59). IEEE. https://doi.org/10.1109/AgileIndia.2012.11

Waterman, M., Noble, J. & Allan, G. (2015). How much up-front? A grounded theory of agile architecture. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1* (pp. 347-357). IEEE Press. https://doi.org/10.1109/ICSE.2015.54

Webster, J. & Watson, R. T. (2002). Analyzing the Past to Prepare for the Future: Writing a Literature Review. *MIS Quarterly, 26(2),* Xiii-Xxiii.

WEF & Accenture (2017). Digital Transformation Initiative. Retrieved March 12, 2018 from: https://www.accenture.com/t20170116T084450__w__/us-en/_acnmedia/Accenture/Conversion-Assets/WEF/PDF/Accenture-DTI-executive-summary.pdf

Woods, E. (2015). Aligning Architecture Work with Agile Teams. *IEEE Software, 32(5)*, 24-26. https://doi.org/10.1109/MS.2015.119

Woods, E. (2016). Software Architecture in a Changing World. *IEEE Software, 33(6),* 94-97. https://doi.org/10.1109/MS.2016.149

Yang, C., Liang, P. & Avgeriou, P. (2016). A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software, 111*, 157-184. https://doi.org/10.1016/j.jss.2015.09.028

Zimmermann, O. (2016). Designed and delivered today, eroded tomorrow? In *Proccedings of the 10th European Conference on Software Architecture Workshops* (p. 7). ACM. https://doi.org/10.1145/2993412.3014339

Zimmermann, O. (2016). Microservices tenets: Agile approach to service development and deployment. *Computer Science - Research and Development, 32(3-4)*, 301-310. https://doi.org/10.1007/s00450-016-0337-0

✉ Correspondence

Zoran Dragičević

Company Boksit
Trg rudara 1, 75446, Milići,
Republic of Srpska, Bosnia and Herzegovina

E-mail: zoran.dragicevic021@gmail.com