Contextual Factors of Architectural Strategy for Complex Systems

Mirjana Maric(✉), Predrag Matkovic, Pere Tumbas, and Veselin Pavlicevic

Faculty of Economics in Subotica, University of Novi Sad, Segedinski put 9-11, 24000 Subotica, Serbia
{mirjana.maric, predrag.matkovic, pere.tumbas, pavlicevic}@ef.uns.ac.rs

Abstract. Architecture is the "backbone" of every software product, regardless of the development process used. However, its role, significance, and development strategies differ from one software development process to another. Traditional architecture development, based on a well-defined architectural process that involves the three following architectural phases–architectural analysis, synthesis, and evaluation–is based on the Big Design Up Front strategy. In agile development, architecture is generated gradually with each iteration, as a result of continuous code refactoring, not some predefined structure. Therefore, agile software development relies on an opposite extreme architectural strategy, emergent architecture.

The research topic of this paper is focused on the development of architecture for modern complex systems, which cannot be based on either of the two aforementioned extreme architectural strategies. Development of an architectural strategy for a complex system is significantly influenced by contextual factors.

This paper presents the results of a qualitative empirical research, carried out on a sample of 20 expert practitioners. The results represent contextual factors that the practitioners–surveyed respondents–consider when deciding to which extent will the emergent strategy be extended with explicit architectural practices typical to the traditional architecture development. Obtained results suggest that agile practitioners scale up agile processes, in terms of architecture development, with regard to the contextual factors of the system being developed.

Keywords: Agile software development · Software architecture · Contextual factors

1 Introduction

Agile development processes are nowadays used by countless software development companies worldwide. The main motivation for the use of agile processes is to reduce costs and increase adaptability to changes resulting from dynamic market conditions [1]. Emergence of agile development processes has had a significant effect on the software industry, but it also opened numerous issues that are currently occupying academic researchers. One of the current questions pertains to the role of software architecture and its significance in agile processes [2, 3].

S. Wrycza and J. Maślankowski (Eds.): SIGSAND/PLAIS 2017, LNBIP 300, pp. 111–123, 2017.
DOI: 10.1007/978-3-319-66996-0_8

Software architecture implies decomposing the system into smaller parts, modules, which are developed independently and can be reused or replaced with other modules. Modularization increases flexibility by creating loose links between highly interconnected parts of a system, localizing the effect of changes made to a particular component; it improves the comprehensibility of the system, easing its maintenance and further development [4].

There are two opposite strategies for developing software architecture, with completely different attitudes to the role and the significance of architecture in the software development process: emergent architecture and Big Design Up Front (BDUF).

BDUF strategy involves a well-defined architectural process, comprised of a vast number of explicit architectural activities and decisions that precede the implementation of system functionalities. In other words, the complete architectural design is done at the beginning of a project. This architectural strategy is typical to the traditional software development.

In contrast, emergent architecture is typical to agile development, guided by the principles of "value driven development" and focused on early delivery of value to the user. Agile processes emphasize the value of early development of functionalities, while seeing the architecture as an outcome of the development process. Agile practitioners often consider architecture development economically unjustified, believing that it does not provide ROI, but rather increases total project costs. In agile processes, concepts of metaphor and refactoring are considered adequate replacements for the traditional architecture development process. More exactly, agile processes do not have typical architecture development activities, such as analysis, synthesis, and evaluation [5]. Architecture is rather developed gradually with every iteration, as a result of continuous changes to the source code, not as a result of a predefined structure [5–7].

However, development of architecture of modern complex systems must be based on a strategy that balances the previously described extremes. It would be highly risky to rely entirely on emergent architecture in complex system development, since it is possible to reach a point where the present architecture cannot be amended through refactoring. Such situation would require a complete redesign of the architecture, which implies a significant increase in costs, along with client dissatisfaction. Although agile processes help developers achieve efficiency, quality, and flexibility in change management, complex system development requires application of explicit architectural practices [8–11]. This helps avoid high amount of refactoring, and reduces the risk of architectural erosion [11–13].

The extent to which explicit agile architectural practices are included in an agile development process depends on the contextual factors of the system. In accordance with the described research subject, the research problem addressed in this paper is the influence of contextual factors on architectural strategies for complex system development.

The research question was defined in line with the defined research problem:

RQ: Which empirically identified contextual factors influence the application of explicit architectural activities and the extent of up-front architectural planning in agile software development processes?

The answer to this research question will be given in an overview of primary qualitative results of the conducted empirical research.

2 Research Methodology

This empirical research was based on qualitative methods. Collection of empirical data was conducted by means of a semi-structured interview. The initial set of questions was based on an analysis of prominent literature related to the research topic. The research instrument was subject to evaluation by a group of experts. Subsequent to expert evaluation, content validity index was computed for each of the questions, as well as for the entire questionnaire, in accordance with the recommendations by Polit and Beck [14]. The content validity index of the first version of the questionnaire was 0.76, which suggested that it needed to be modified in accordance with expert's suggestions. Modifications involved elimination of questions with values of the content validity index below 0.8, amendment of how certain questions were formulated, as well as merging several questions into single questions.

The interviews were conducted face-to-face, recorded, transcribed and submitted to interviewees for verification. Research results were obtained through thematic content analysis. Coding of interview data and the thematic content analysis were carried out in the NVivo software suite, following recommendations by Miles and Huberman [15]; all interview transcripts were studied several times prior to the development of the initial list of codes, which was based on the relevant literature; new codes were introduced inductively throughout the analysis. Codes were grouped into categories based on similar characteristics and subsequently organized into clusters.

The nature of the research necessitated purposive sampling ($n \geq 20$). Therefore, the sample consisted only of experts with significant experience (minimum 5 years) in complex system development with agile processes and software architecture development, selected from prominent Serbian IT companies. Since the study was limited to one country, the authors purposely included respondents outsourced by companies from various countries, as well as ones employed by global software development organizations. Bootstrapping with 1000 replications was carried with the aim to increase the stability of the findings.

3 Contextual Factors of Software Development

Most of the software development processes commonly recognized as "agile", such as Scrum and XP, have now been around for more than two decades. The term "Agile" denotes iterative and incremental approach to software development, which relies on the principles proclaimed in the Agile Manifesto [16]. Over the years, there has been much interest in the architectural implications of applying agile processes in large-scale projects. Kruchten [17] introduced the term "Agile Sweet Spot" to designate the conditions from which the agile methods stemmed, are where they are most likely to succeed. The author states that out-of-the-box agile methods are most suitable for small, collocated groups of 10–15 people (which allows for good communication), sharing a common culture and working in close relations with the customer on a new ("green field") software project with s short lifecycle. As argued by Abrahamsson et al. [2], a project in such settings does not require many architectural activities. However, as the

project context moves from the "agile sweet spot", the process needs to embrace certain architectural practices, suited to the project context.

Several authors have made efforts to classify the contextual factors of software development projects. In a paper concerning the applicability of agile development practices in projects with different characteristics and backgrounds, Kruchten [18] presented empirically identified contextual factors of software development, divided the into two levels: (1) organizational level (environmental) factors, and (2) project-level factors. The organizational level factors include the business domain, number of instances of the software system to be deployed, maturity of the organization, and the level of innovation. Project-level factors (influenced by organizational level factors) include: system size, stable architecture (whether there already is a set of commonly used patterns), business model, team distribution, rate of change, age of system, criticality, and governance. The latter constitute what the author referred to as the "octopus model" of key contextual factors of software development, depicted in Fig. 1. Such classification of factors enables teams to carry out an analysis of the project context prior to its initiation and determine which practices or methods are unsuitable, and which are essential.

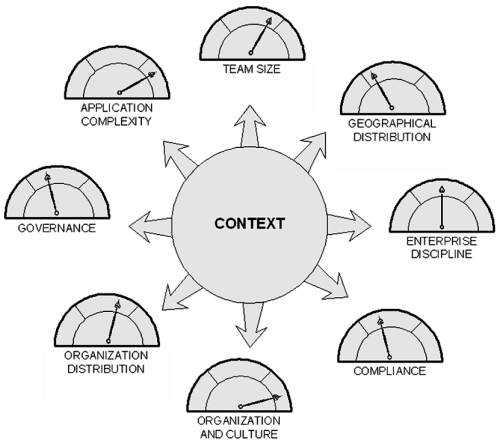

Fig. 1. Contextual factors for agile software development (adapted from [18])

Similar to Kruchten et al. [18], Ambler [19] composed a set of scaling factors that influence the efforts of applying agile and lean processes to large projects: (1) team size – from under 10 developers to hundreds of developers; (2) geographical distribution – from co-located to globally distributed; (3) compliance – from low risk to critical/audited; (4) organization and culture – from open to entrenched; (5) organization distribution – from in-house to third party; (6) governance – from informal to formal; (7) application complexity – from simple, single platform to complex, multi-platform; (8) enterprise discipline – from project focus to enterprise focus. These factors are intended as a reference in the analysis of the context prior to tailoring the development process to suit an individual project.

Boehm et al. [20] have devised a set of guidelines for finding the right balance between agility and architecture in the type of projects that require both. They focus on three contextual factors for determining how much agility and architecting are enough in a software development project [21]: project size, criticality, and volatility. Babar et al. [22] assumed that the three factors proposed by Boehm et al., coupled with other possibly influential contextual factors, also determine whether a satisfactory architecture can emerge from continuous refactoring in an agile project. Their empirical research resulted in a framework with 20 contextual factors, subsequently grouped into four distinct categories: project, team, practices, and organization.

As the project domain shifts from the "agile sweet spot", low ceremonialism and high interactivity, the key characteristics of agile processes, seem to increase the risk of project cost proliferation, architectural erosion, or even project failure. In the following text, we discuss the results of the empirical research, where we aim to identify contextual factors used by practitioners to determine the extent of up-front architectural analysis and design necessary for a particular project.

4 Research Results

Qualitative analysis of the empirical data through inductive text coding resulted in key categories, which served as a structure for the presentation of research results. The main category, titled "Agile Architecture Design" concerns the very process of practical agile architecture development. This category comprises three subcategories, namely: "Factors od Architectural Strategy", "Explicit Architectural Activities", and "Roles and Responsibilities in Architectural Decisions". All of the subcategories influence the development of agile architecture of complex systems in a specific sense; however, in line with the research questions, this paper will be focused on the category "Factors of Architectural Strategy".

The category "Factors of Architectural Strategy" contains numerous concepts, further divided into four subcategories:

1. System complexity: complexity of the problem being solved, complexity of the problem domain, complexity of the architectural solution, complexity of requirements
2. Requirements: quality and volatility
3. Team characteristics: experience, communication, knowledge, team size, etc.
4. Stakeholders: agility, quality of communication, knowledge, etc.

The subcategories listed above represent empirically identified contextual factors that agile practitioners take into consideration when developing architectural strategies for complex system development. Architectural strategies for complex systems always fall between the two extremes: emergent architecture, and BDUF. Which extreme will it be closer to depends on the contextual factors, which are presented and clarified in the following text, along with the interviewees' responses (RSP 1-20).

4.1 System Complexity

System complexity is determined by three main attributes: scale (number of things within a system), diversity (variety of things in a system), and connectivity (number of relationships between the things) [23]. Agile teams have explained complexity through the following terms: complexity of the problem being solved, complexity of the problem domain, complexity of requirements and the architectural solution.

Respondents believe that the complexity of the problem being solved and the complexity of the problem domain largely influence the length of the planning phase, especially in cases when the problem/domain is vast and cannot entirely be comprehended by a single person, but requires brainstorming with multiple team members, who are experts in their respective areas (RSP16).

Complexity of the architectural solution has a significant impact on the architecture development strategy as well. It determines the scope of architectural decisions that need to be made prior to implementation of system functionalities. This factor is most influential in mission safety critical products, where even the slightest problems with system performance can cause disastrous consequences.

In practice, greater complexity of requirements also increases the scope of up-front architectural decisions, which moves the architectural strategy towards the BDUF extreme. Complex system requirements can include cross-platform compatibility, integration with a legacy system, alignment with the legislation, or fault tolerance. These requirements represent system constraints and imply a greater extent of architectural analysis and planning at the beginning of a project, each release, or even a sprint (RSP14).

Requirement for integration and interaction with legacy systems demands spending considerable time on the analysis of legacy systems and the technology used for their development. The challenge of understanding legacy systems becomes even greater if the documentation of the legacy system is sparse, or even non-existent. Similarly, finding the best way to interact with the legacy system is no less of a challenge (RSP1, RSP19). Respondents stated that the biggest problem with this type of requirements is to resolve all issues originating from legacy data, i.e., to determine which data is relevant to the development of the new system's data model, where the legacy data will be stored, and how they would be managed until the software is in production. Transition from a legacy system is a secondary project that should be planned within the development project (RSP15).

According to the respondents' experience, the requirement of cross-platform compatibility has a vast influence on the architectural solution. Therefore, it necessitates in-depth analysis at the beginning of the project, aimed at selecting adequate technology. The amount of time spent on determining the right technology greatly depends on the software architect's knowledge and experience. Choice of technology is an important moment in architecture development, since not all technologies are equally suitable for solving particular problems and developing certain types of software products (RSP3, RSP5, RSP20).

The complexity of requirements is also determined by the number of users, number of transactions, as well as the amount of data that the system is expected to process. These requirements are directly related to scalability, and therefore must be thoroughly considered prior to implementation (RSP13, RSP17, RSP2).

It is interesting that the respondents did not directly relate project size with complexity, which contravenes the findings in the literature [2, 24, 25]. The respondents believe that the fact that a project requires a great amount of time, and/or participation of many people does not imply a greater extent of architectural planning at the start. A large-scale project may just require a lot of "manual labor" on coding and the development of the solution (RSP12). The respondents associated the scale of a project with project management challenges, which can purportedly be mitigated through service or microservice architectural solutions (RSP16).

The described empirical results point to the conclusion that the respondents directly link the factors of complexity with the extent of up-front architectural decisions necessary in architecture development. In other words, the greater complexity of a system being developed will cause the emergent architecture to be based on a greater extent of up-front architectural decisions. It should also be noted that architecture development essentially remains incremental, eliminating the possibility of falling into the trap of BDUF. The respondents stated that the time necessary for up-front architectural decisions can significantly be reduced with the use of a reference architecture (predefined architectural solutions). Such respondents' opinions are in line with the findings in the literature [26, 27], which suggest that the use of a reference architecture improves the agility of the development process.

System requirements, especially their volatility and insufficient quality, significantly influence the choice of an architectural strategy and its positioning in relation to one of the two extremes.

4.2 Requirements

Most of the respondents stated that projects are often realized in unstable conditions, since the clients are not entirely sure what they need at the beginning of a project. They mostly come with an idea that they had not thoroughly elaborated, which can result in wearisome and lengthy identification of architectural requirements that can delay the implementation phase. In addition to that, clients are often unable to clearly formulate their requirements, which is yet another cause of delay (RSP8, RSP10, RSP6).

Quality of requirements is in direct proportion to the amount of time a software architect needs to spend on up-front architectural analysis at the beginning of a project. Identification of architecturally significant requirements at the beginning of the project is beneficial not only to the development team, but to the client as well. Architecturally significant requirements serve as the basis for determining the scope of the future system, and are crucial to the design of the architecture for the principal part of the software (RSP8, RSP4). This is where non-functional requirements, of which the clients are mostly unaware, are of utmost importance. For this reason, the software architect should help their clients identify non-functional requirements and discuss the possibilities of their implementation, since some non-functional requirements may be mutually exclusive (RSP3, RSP4).

Initial definition of architecture involves identification of elements that are costly to develop, and therefore should not undergo changes throughout the project. Subsequent identification or modification of these crucial architectural raises the question whether to develop an entirely new architectural design, or to attempt refactoring the present

unsatisfactory solution. In both cases, costs increase enormously, as well as the duration of the project (RSP8).

In order to counteract these challenges, practitioners often use the concept of spikes. Spikes are used in two different cases: when the clients have a certain requirement, but are unsure what they want in terms of functionalities, and when the software architect and the development team are unsure how to implement certain requirements. In these cases, practitioners most frequently opt for developing a time-boxed architectural prototype.

Volatility of requirements is the second most common problem software architects and agile teams face. Volatility of requirements is associated with the implementation stage, when the client states changes to their expectations from the software. This can severely jeopardize the viability of the developed architecture (RSP13, RSP12).

This risk can be mitigated through additional efforts at the beginning of the project, aimed at reaching a consensus with the stakeholders on the business and architectural vision. This does not involve a detailed elaboration of all the requirements, which would not even be possible in the modern-day business environment, but rather focusing on architecturally significant requirements for the main part of the system. Nowadays, software is crucial to the success of a business. In line with this, software development is a continuous process, as is any business process within an enterprise. Changes in business instigate new requirements and continuous adaptation of the software solution, which implies that architecture development must be a continuous, iterative and incremental process. The architecture initially developed for the main part of the system evolves throughout the project, while the detailed elaboration of a set of requirements takes place during the planning stage of an iteration in which they will be implemented. This lead to a conclusion that numerous architectural decisions must be postponed until the requirements are well understood, which can be called Just-In-Time architectural planning (RSP6, RSP9).

Changes to the initially set architecture is justifiable only in case of a great advance in the client company's operations and a much more extensive use of the system than the intended, when the performance of the present architectural solution acts as a bottleneck (RSP8).

These results suggest that the respondents believe that there is a direct link between the quality and volatility of requirements and the length of the architectural planning stage. Quality of requirements influences the amount of time a software architect needs to spend on up-front architectural analysis, in order to determine the scope of the system and architecturally significant requirements, which serve as the basis for the architecture of the main part of the software.

4.3 Stakeholders

According to the respondents, stakeholders' qualities represent an important factor of architectural strategy definition. Their openness to continuous collaboration through active participation in the project and delivery of feedback to the project team influences the choice of architectural strategy. Respondents' views are in line with the findings in the literature; specifically, Friedrichsen (2014) pointed that the extent of

up-front analysis and design depends on software architect's experience, skills, knowledge, as well as good communication with the stakeholders.

In addition to stakeholders' openness to active collaboration throughout the project, the quality of their involvement and communication with them is also an important factor. Stakeholders should understand the problem, and also be able to formulate and express their requests and needs (RSP4, RSP13).

A great portion of problems in a project, including those associated with software architecture, originates from stakeholders' lack of competence to provide adequate requirements. Therefore, for the success of the software architecture and the whole project, it is of utmost importance to identify key stakeholders, i.e. the stakeholders with the greatest influence on the software being developed, at the start (RSP7, RSP8, RSP13).

The choice of an architectural solution should be a result of a close collaboration between the software architect and the stakeholders. The software architect's role is to present stakeholders with alternative architectural solutions and descriptions of expected results each of the solutions would provide (RSP3), along with the costs associated with each solution (RSP8). The stakeholder's role is to make the final decision on the course to be taken in the architecture development (RSP3).

It is challenging, even impossible, to follow an architectural strategy close to emergent architecture when working for a non-agile stakeholder (client/user). Stakeholders often refuse to accept the principles of agile development, primarily because they want to "insure" themselves by defining the scope, timeline and quality in the contract. If all this is preset, it is not possible to develop software with full adherence to agile principles (RSP9). This situation is particularly evident in complex system development projects, where clients tend to have a negative attitude towards fully agile development. This is principally because of the elimination of the planning stage and emergent architecture, distinctive of agile processes (RSP8).

Based on all above, it can be concluded that stakeholders' qualities are a factor that influences the extent of architectural planning in complex system development. The results suggest that stakeholders do not support architectural strategies based entirely on emergent architecture.

4.4 Team Characteristics

Results within this category suggest that characteristics of a development team, such as team size, understanding of the problem being solved, knowledge in the problem domain, familiarity with technologies and current trends in architectural options, experience, and quality of communication and collaboration influence the development of complex systems' architecture with agile processes

If the team was involved on solving a similar problem before, then they possess relevant experience and knowledge, which reduces the time and effort required for up-front architectural analysis and design. This implies that software architects do not need to develop prototypes or organize brainstorming sessions, since they can decide on the architectural solution based on their experience (RSP11).

Team member's familiarity with the problem domain, as well with the technology to be used in the development of a solution also reduce the time and effort required for

up-front architectural analysis and design (RSP9, RSP3). If the team is composed of individuals with experience with different alternative technologies and familiarity with their capabilities and limitations, up-front architectural planning and the choice of technology will be accelerated. Nowadays, there are countless free components available, which enables faster development and lower costs (RSP2, RSP5). However, situations where clients impose their technology stack are frequent, especially in outsourcing projects. In such situations, the team requires additional time to explore the technology, as well as to consult with the individuals of enterprises with experience in using these technologies. All of this delays the implementation of functionalities (RSP14, RSP3, RSP18).

In addition to that, time invested in up-front architectural activities will be significantly shorter if the team is composed of individuals with proper knowledge and experience in the domain of software architecture. Experienced software engineers can make a significant proportion of architectural decisions unaided by the software architect, which increases the agility of the team. The younger and less experienced team members are, the more time the software architect needs to spend on up-front architectural planning and design, as to trace out the developers' work as much as possible and mitigate the risk of taking a wrong turn in the development (RSP16, RSP5).

In order to gradually reduce the time and effort the software architect needs to invest at the beginning of the project and during the implementation of the solution, it is necessary to commit to continuous advancement of all team members' knowledge. This practically means that all team members should attend software architects' meetings and actively participate in architectural planning and development of architectural strategies. In addition to that, they should undergo courses such as "professional scrum developer", etc. Education and communication are, therefore, means of enabling developers to unassistedly make architectural decisions during the implementation stage (RSP14).

Such respondents' views correspond to the findings in the literature. Boehm and Turner [20] have concluded that agile development requires a critical mass of experts. The authors defined experts as the members of the team "Able to revise a method (break its rules) to fit an unprecedented new situation." An expert should not blindly follow the rules and instructions, but to have enough knowledge, skills and experience to be able to make decisions based on intuition. Blair, Watt i Cull [28] highlighted the necessity of close cooperation between the software architect and the team, with continuous sharing of ideas and knowledge throughout the entire project.

Most respondents have recognized the number of teams/individuals involved in a project as another factor influencing the extent of up-front architectural planning, particularly owing to the necessity of analyzing dependencies among teams (RSP4). If the components being developed separately by different teams are interconnected, this implies a completely different approach to architectural planning and coordination, as well as to implementation and testing.

Appropriate communication and collaboration is the only way to reduce the need for greater up-front planning in case of a large number of teams.

Respondents' opinions on this issue correspond with the conclusions of Coplien and Bjornvig [29], who stated that architectural efforts can radically be reduced through

good communication between team members, also providing an example of reduction from six months to two weeks.

However, agile collaboration, good communication, and experience are things that agile teams cannot achieve overnight, but rather require that team members spend time working together on various projects. The longer the team members spend together, less time is required for up–front architectural planning, since there are fewer problems in the communication between the developers and software architects. In such cases, it is sufficient to agree on the patterns to be used, discuss the specificities of particular aspects of development, and identify critical relations at the very beginning of a project. The goal is to elevate developers' skills and knowledge in the domain of software architecture, making them able to "tailor" parts of software by themselves, without detailed instructions by an architect (RSP19).

This is in line with empirical findings of Hoda [30], who determined that it requires a certain time for members of a team to develop experience, self-organization, self-evaluation, and self-improvement.

In addition to that, an agile team, in the real sense of the word, implies members sitting in the same room and communicating face-to-face on a daily basis. This further means that agile processes were intended for in-house development projects. However, the respondents stated that most of their projects involve outsourced work, which is the principal reason that these projects can not entirely rely on agile principles and values, but rather need to be scaled through inclusion of explicit architectural activities (RSP14).

5 Conclusion

Research results suggest that agile teams involved in the development of complex systems do not use either of the two extreme strategies for developing software architecture–BDUF and the emergent architecture–but rather employ strategies that fall between the two extremes. Contextual factors of the system being developed determine to which extreme the selected architectural strategy will be closer. This is a matter of establishing a balance between the tactical level of software development, one that provides rapidly visible value through the development of functionalities, and the strategic level, which produces long-term value through the design of software architecture.

Research results point to a conclusion that empirically identified contextual factors (system complexity, system requirements, stakeholders' qualities, and team characteristics) influence the time required for architectural planning in complex system development. Even though this phase is almost entirely eliminated in typical agile projects, results of this research indicate that this is not the case in the development of systems, which corresponds with the views found in previous studies. The reason behind this lies in the fact that agile processes were not initially intended for complex system development, and therefore emergent architecture is an inadequate strategy when this type of software is concerned. In order to be applied in complex system development, agile processes need to be scaled up through inclusion of explicit architectural activities, typical to the traditional architecture development. In other

words, the authors of this study recommend extending the outreach of architectural planning beyond the current sprint, as to prevent loss of flexibility and degeneration of the design. The scope of explicit architectural activities carried out at the beginning varies from project to project, since the contextual factors that influence them are unique for each system. Therefore, the authors recommend a detailed analysis of contextual factors of the future system be carried out prior to the definition of an architectural strategy.

It can be concluded that the development of a complex systems' architecture based entirely on emergent architecture may not necessarily result in "agile architecture". The term "agile architecture" designates an architecture designed in a way that it can easily be changed, i.e., that it can react to the changes in the environment. Therefore, the authors recommend retaining an iterative process for developing complex systems' architecture, but one comprising an adequate number of explicit architectural practices, executed Just-In-Time, and in line with the contextual factors of the future system.

References

1. Ambler, S.W., Lines, M.: Disciplined Agile Delivery, 1st edn. IBM Press, Boston (2013)
2. Abrahamsson, P., Babar, M.A., Kruchten, P.: Agility and architecture: can they coexist? IEEE Softw. **27**(2), 16–22 (2010)
3. Pérez, J., Díaz, J., Garbajosa, J., Yagüe, A.: bridging user stories and software architecture: a tailored scrum for agile architecting. In: Babar, I., Brown, M.l., Mistrik, A.W. (eds.) Agile Software Architecture: Aligning Agile Processes and Software Architectures, 1st edn., pp. 215–241. Elsevier, New York (2014)
4. Parnas, D.L.: On a 'Buzzword': hierarchical structure. In: Proceedings of the IFIP Congress 1974, pp. 336–339 (1974)
5. Babar, I., Brown, M.l., Mistrik, A.W.: Agile Software Architecture Aligning Agile Processes and Software Architectures, 1st edn. Elsevier, Waltham (2014)
6. Beck, C., Andres, K.: Extreme Programming Explained: Embrace Change, 2nd edn. Addison Wesley, Boston (2004)
7. Thapparambil, P.: Agile architecture: pattern or oxymoron? Agil. Times **6**(1), 43–48 (2005)
8. Babar, M.A., Abrahamsson, P.: Architecture-centric methods and agile approaches. In: Proceedings of the 9th International Conference on Agile Processes and eXtreme Programming in Software Engineering, pp. 238–243 (2008)
9. Parsons, R.: Architecture and agile methodologies—how to get along. In: WICSA (2008)
10. Nord, R.L., Tomayko, J.E.: Software architecture-centric methods and agile development. Softw. IEEE **23**(2), 47–53 (2006)
11. Ihme, T., Abrahamsson, P.: The use of architectural patterns in the agile software development on mobile applications. In: ICAM 2005 Internetional Conference on Agility, vol. 8, pp. 1–16 (2005)
12. Stal, M.: Refactoring software architectures. In: Agile Software Architecture: Aligning Agile Processes and Software Architectures, pp. 130–152. Elsevier (2014)
13. Kruchten, P.: Situated agility: context does matter, a lot. In: 9th International Conference on Agile Processes and eXtreme Programming in Software Engineering (2008)
14. Polit, C.T., Beck, D.F.: The content validity index: are you sure you know what's being reported? critique and recommendations. Res. Nurs. Heal. **29**, 489–497 (2006)

15. Miles, M.B., Huberman, A.M.: Qualitative Data Analysis: An Expanded Sourcebook, 2nd edn. Sage, Thousand Oaks (1994)
16. Beck, K., et al.: The Manifesto for Agile Software Development (2001)
17. Kruchten, P.: Scaling down large projects to meet the agile 'sweet spot'. Ration. Edge, 1–14, August 2004
18. Kruchten, P.: Contextualizing Agile Software Development, pp. 1–12 (2010)
19. Ambler, S.W.: Agility@Scale: Strategies for Scaling Agile Software Development. Agile and domain complexity (2010)
20. Boehm, R., Turner, B.: Balancing Agility and Discipline: A Guide for the Perplexed, 1st edn. Addison-Wesley, Boston (2003)
21. Boehm, B., Lane, J., Koolmanojwong, S., Turner, R.: Architected agile solutions for software-reliant systems. In: Agile Software Development: Current Research and Future Directions, pp. 165–184. Springer, Heidelberg (2010)
22. Chen, M.A., Babar, L.: Towards an evidence-based understanding of emergence of architecture through continuous refactoring in agile software development. In: 2014 IEEE/IFIP Software Architecture (WICSA), pp. 195–204 (2014)
23. Kruchten, P.: Complexity made simple. In: Proceedings of the Canadian Engineering Education Association (2012)
24. Boehm, R., Turner, B.: Using risk to balance agile and plandriven methods. IEEE Comput. **35**(6), 57–66 (2003)
25. Cockburn, A.: Agile Software Development: The Cooperative Game, 2nd edn. Addison Wesley, Boston (2007)
26. Mirakhorli, J., Cleland-Huang, M.: Traversing the twin peaks. IEEE Softw. **30**(2), 30–36 (2013)
27. Kruchten, J., Obbink, P., Stafford, H.: The past, present, and future for software architecture. IEEE Softw. **23**(2), 22–30 (2006)
28. Blair, S., Watt, R., Cull, T.: Responsibility-driven architecture. IEEE Softw. **27**(2), 26–32 (2010)
29. Coplien, G., Bjornvig, J.O.: Lean Architecture for Agile Software Development, 1st edn. Wiley, Cornwall (2010)
30. Hoda, R.: Self-Organizing Agile Teams: A Grounded Theory. Victoria University of Wellington, Wellington (2010)