STRATEGIC MANAGEMENT, Vol. 21 (2016), No. 1, pp. 016-022

UDC 72:004.42

Received: June 22, 2015
Accepted: September 17, 2015

The Role of the Software Architect in Agile Development Processes

Mirjana Marić
University of Novi Sad, Faculty of Economics in Subotica, Subotica, Serbia
Pere Tumbas
University of Novi Sad, Faculty of Economics in Subotica, Subotica, Serbia

Abstract
Software architecture stands as the backbone of any software system, regardless of the methodology it was developed with. The responsibility for the choice of the architectural solution, its design and evaluation in the traditional development process lies in the role of the software architect. Traditional architecture development is based on three architectural phases: architectural analysis, architectural synthesis and architectural evaluation. The highly ceremonial character of the traditional process is also reflected in the role of the software architect, who is responsible for generating numerous architectural artefacts.
The proponents of agile development start from the position that an architecture emerges gradually, after every iteration, as a result of continuous code refactoring, rather than from a previously built structure. In accordance with this, agile processes do not contain any of the traditional phases (the analysis, the synthesis and the evaluation) of the architecture development process. Agile processes also have a different view of the responsibility for a developed architectural solution, delegating it to all the members of the agile team.
In terms of expanding agile development processes with traditional architecture processes, with the aim of enabling the development of a complex system, the current research and efforts have also identified change in the role of the software architect. This paper aims to describe the modified role of the software architect in agile processes for developing complex software solutions, integrating findings of the existing literature and the results of the conducted empirical research. The empirical research results feature as a part of the research conducted within the doctoral dissertation entitled "The Methodological Framework for Developing the Software Architecture of Business Software in Agile Processes", submitted at the Faculty of Economics in Subotica of the University in Novi Sad.

Keywords
Software architect, software architecture, agile development processes.

Introduction

Advocates of the postulate of the vital role of an architecture in the development of high-quality software intensive systems question the scalability of agile development processes, as they do not pay sufficient attention to architecture (Ihme & Abrahamsson, 2005; Nord & Tomayko, 2006; Parsons, 2008). They tend to characterise agile practices as amateurish and limited to the domain of very small internet business solutions.

Contrary to this, proponents of agile development processes identify a previously planned architecture development with the traditional Big Design Up-front (BDUF) strategy, which is contrary to the basic agile principles and values, as it leads to massive documentation and the development of functions that are unnecessary in implementation (Babar, 2014). Agile development advocates a philosophy based on rapid development, direct and continuous communication with stakeholders, the team self-organisation and the creativity of all participants, with the aim of avoiding excessive planning, modelling and documenting (Abrahamsson, Babar, & Kruchten, 2010). Agile processes do not have activities typical of the development of software architecture, such as analysis, synthesis and evaluation, as these require additional costs without producing an adequate business value for the client (Babar, 2014). For

this reason, they are focussed on developing software functionalities, regarding an architecture as a by-product of the development process itself. They consider the concept of metaphor and the refactoring technique as an adequate substitute for the traditional process of an architecture development. According to them, an architecture emerges gradually, after each iteration, as a result of continuous modifications of the program code (an emergent architecture) rather than from a previously built structure (Beck, 2004;Thapparambil, 2005; Babar, 2014). Such a development strives to shorten as much as possible the initial development phase, which, according to the proclaimed agile principles, has no visible and useful results for the user.

Contemporary software intensive systems are characterised by a high level of decentralisation, heterogeneity and integration with other systems. In accordance with this, it can be said that, today, there is a new gap in the software industry between the constant growth of business software and shortcomings of agile methodologies to support its development and maintenance. The described state has in practice initiated a large amount of research over the past year, which is on the line of finding a compromise between both the rapid delivery of the functional product to the user and the development of a stable software solution (Nord & Tomayko, 2006; Parsons, 2008; Ihme & Abrahamsson, 2005; Bellomo, Ozkaya, & Nord, 2013). Such a trend can be characterised as the traditionalization of agile development processes (Matković, Tumbas, & Sakal, 2011).

Over the past few years, the scientific community has recognized that opposing development processes (traditional vs. agile) have been playing complimentary roles in the development of software and has studied the possibility of their coexistence and advantages of their integration (Nord & Tomayko, 2006;Kruchten, 2007; Ambler & Lines, 2013). Agile processes provide companies with efficiency, quality and flexibility in adopting change, but it is important for the development of complex software solutions to use some of the traditional architectural practices as well (Nord & Tomayko, 2006; Parsons, 2008; Ihme & Abrahamsson, 2005; Babar & Abrahamsson, 2008). Current research shows that the refactoring technique, as the dominant architectural practice in agile processes, can be a sufficiently successful practice, as long as high-level software architecture is good. This is the way how to avoid a high degree of refactoring, implicating high system development costs in later development phases, as well as the erosion of an architecture, which can jeopardise the success of the entire process (Ihme & Abrahamsson, 2005; Kruchten, 2008;Stal, 2014).

Craig Larman opines that the described dichotomy between agility and an architecture is unfounded. *Along these lines, Satoshi Basaki noted,* "It *seems* that *many agile method users misunderstand* what *agile methods* are, *just ignore architecture*, and *jump* onto refactoring" as the one and *only* panacea. The cornerstone of this claim is said to be in the words of the creator of the XP agile methodology, Kent Beck, who proposes: "Architecture is just as important in XP projects as it is in any software project. Part of the architecture is captured by the system metaphor." Beck also pointed out the significance of dealing with non-functional system requirements, as well as the possibility to extend the XP process development, to the extent that depends on the project context (Abrahamsson et al., 2010).

Current research confirms the importance of finding "middle-of-the-road" between two extreme architecture development methods (Kruchten, 2007; Babar & Abrahamsson, 2008; Boehm, 2002), with a focus on finding mechanisms for bridging the gap (Nord & Tomayko, 2006; Parsons, 2008; Ihme & Abrahamsson, 2005; Lycett et al., 2003; Boehm & Taylor, 2005; Babar, 2009; Abrahamsson, Babar, & Kruchten, 2010; Babar, Ihme, & Pikkarainen, 2009).

In the context of the described research subject, the following research questions emerge:

RQ1: What is the role of software architecture in the development of complex software solutions by agile development processes, according to current findings in the literature?

RQ2: What do the results of the conducted empirical research show in terms of the role of the software architect?

1. Research methodology

The empirical research was conducted by the application of the classical variant of the Delphi technique, implying three "circles" or research iterations (Helmer & Rescher, 1959; Keeney et al., 2011).

The first iteration was the qualitative research component and the data were therefore gathered by means of semi-structured interviews. A semi-structured interview consisted of a previously prepared and expert-evaluated set of questions.

The interviews were conducted face-to-face and recorded in order to obtain the greater accuracy and completeness of the gathered data. The nature of the research problem required a deliberated selection of sample units ($n \geq 20$). The sample was, therefore, comprised solely of experts versed in agile development and software architecture design. The research was conducted in relevant companies in the ICT sector in Serbia. The previously transcribed data were subjected to a qualitative analysis in the NVivo software. The qualitative analysis of the data was conducted according to the recommendations of Miles and Huberman (1994).

The second iteration of the empirical research was the quantitative component. The research instrument was a questionnaire with checklists, check tables, and the Likert-formatted evaluation scale. It was made based on the results of the first iteration and the respondents received it in the electronic form (e-questionnaire). A quantitative analysis of the gathered data was performed in the SPSS software.

The third iteration of the research was also performed by the electronic dissemination of the questionnaire. The questionnaire was made based on the results obtained in the second circle of research.

The following methods were used for the quantitative analysis of the data in the second and the third circles of the research: the measures of the central tendency/location and dispersion measures – the median and the interquartile range; the arithmetical mean and the standard deviation; the coefficient of variation; Spearman's rank correlation coefficient; the *Kendall rank correlation coefficient; Kendall's coefficient of concordance (W);the Goodman-Kruskal Index of Predictive Association (for assessing the degree of stability between two consecutive Delphi iterations); kappa; the Content Validity Index; inferential statistics procedures, the chi square test; bootstrapping; a quantitative content analysis; etc.*

The theoretical research was conducted by applying the Systematic Literature Review Method. The Systematic Literature Review was realised according to the framework set by Kitchenham (2004): Planning the Review, Conducting the Review, Reporting the Review. The stages associated with *Planning the Review*are: 1. The identification of the need for a review; 2. The development of a review protocol. The stages associated with *Conducting the Review*are: 1. The identification of research; 2. The selection of primary studies; 3. A study quality assessment; 4. Data extraction & monitoring; 5. Data synthesis. *Reporting the Review*is a single stage phase, whose presentation was given in the Theoretical Background Section.

The defined research protocol implied a strategy for searching the primary research material. Specifically:

- The search sources were chosen – the electronic databases IEEE Xplore, Science Direct, and the ACM Digital Library.
- The search keywords were defined: agile software architecture, agile methods (methodologies) and architecture, agility and architecture, software architect.
- The criteria were defined for the inclusion and exclusion of the research material – the acceptable items were peer-reviewed scientific and expert papers published in periodicals and collections of papers presented at conferences and workshops from 2000 to 2014, and all the papers were excluded from the analysis if their term of agility had no link to agile methodologies, the papers without empirical research or a proposed approach/method, and the papers based solely on experts' opinions.
- The quality of the research material was evaluated and met the defined criteria of inclusion, according to the criteria proposed by Dyba and Dingsoyr (2008),and the key data from the relevant research material were extracted and synthesised with the NVivo software, for the easier management of the key concepts, findings and conclusions contained in the publications.

The total number of hits, for each electronic basis individually, is presented in Table 1. The conducted research resulted in ten relevant papers dealing with the issue of the role of the software architect in agile development processes.

Table 1 Search results of electronic databases

Data sources	Number of hits by keywords	Number of papers Included in further analysis	Number of excluded papers
IEEE Xplore	701	43	658
Science Direct	46	12	34
ACM Digital library	237	12	225
Total	984	67	917

2. Research results

The following section of the paper describes the role of the software architect in agile processes of developing complex software solutions, by using findings from the existing literature and the results of the conducted empirical research.

The empirical research results show that a formal role is played by the software architect, who is also a very experienced programmer, in all agile times. The findings are in compliance with the opinions and proposals of (Coplien & Bjornvig, 2010). As an experienced programmer, the software architect usually joins the development team at the beginning of a project, so as to set the main part of the software (RESP.5) with programmers. In agile processes of developing complex software solutions, the role of the software architect is essentially changed in comparison with the traditional one, demanding his involvement throughout the development process. The software architect features as some kind of a mentor, advising and helping programmers during the project in resolving architectural questions and problems (RESP.11). In practice, such a condition is in accordance with opinions expressed in the literature, proposed by (Hadar & Sherman, 2012).

Faber (2010) believes that architects should provide value to clients, through meeting non-functional demands of the system and providing continued support to programmers during the process of implementing solutions. Agile teams are aware of this fact in practice and strive to set up an ideal situation, where architects would feature as service providers to programmers and the client. However, achieving such a role of the software architect inevitably requires "raising awareness, confidence, skills and knowledge in the domain of the problem and technology" of all team members. An identified problem in practice is a constant inflow of new people in the fact that an average engineer in a team is at the junior level (RESP.20). Raising the level of the technical knowledge of team members is best achieved by their involvement in discussions on architecture during its setup at the beginning of the project and during planning iterations. This is the way how to avoid bottlenecks of people who become specialists in architecture, whereas other team members lack both architectural knowledge and an overall picture of a solution (RESP.20).

Respondents believe that architect themselves must build (on) their system-related, technological and domain knowledge at the very beginning of the project so that they could provide continuous assistance to programmers and stakeholders (RESP.19). For these reasons, it is essential for architects to participate in meetings with stakeholders and hear first-hand what the problems are because, on the contrary, they will never obtain full information through the documentation forwarded to them (RESP.7), and, although being technical personnel, they are still not at the level of architects, so they are unable to communicate the full 100% of all requirements (RESP.16).

Hopkins and Harcombe (2014) deem that the success of software architecture requires that software architects view the problem being solved at the beginning of the project, and do so from several different perspectives, as each business problem is different and has unique architectural aspects of its own. The respondents have a similar opinion, stating that the most important thing for an architect in resolving a problem is to understand how the target organisation operates (RESP.13). Understanding the business processes, i.e. the business case, of the target organisation is the basis for identifying architecturally significant requirements (RESP.12).

Developing complex systems requires the existence of a formal role in the team, involved in the identification of requirements (the product owner), which, on the other hand, does not mean that the software architect should not participate in their prioritisation (RESP.16), either. On the contrary, the respondents' opine that the product owner should prioritise requirements from the aspect of the user value, and the architect from the aspect of the costs, risks and technical dependencies of the solution (RESP.20). Developing an agile architecture requires generating a unified list of prioritised functional and non-functional system requirements (RESP.16).

Such an opinion of agile teams can in practice be said to be in line with Madison's (2010) view of the role of the software architect, who should set up a balance between business and architectural priorities so as to produce an agile architecture.

Blair, Watt and Cull (2010)opine that close collaboration between the software architect and the team is the key to the success of the project as a whole. In this regard, Kruchten (2013) especially points out the importance of collaboration between the architect and the business analyst (in terms of the identification and prioritisation of demands), the project manager (with respect to delivery plans, risks and costs) as well as programmers (concerning issues of the implementa-

tion of the limitations and evaluation of prototypes). The RESP.19 respondent links the success of an architecture to collaboration between the architect and the product owner and programmers. He also points out that should any of these roles fail, that can be disastrous for the architecture, because if the product owner does not understand well what he/she should do, a small omission in architecture by the architect may result in months of additional work. Likewise, an inadequate quality and unmotivated programmers may jeopardise the sustainability of the architectural solution (RESP.19).

Babar (2009)emphasises the importance of an interaction between the software architect and clients, with the aim of identifying requirements and their prioritisation and generating the artefact of the Software Architectural Overall Plan (SAOP). Buschmann (2012) points out the active participation of all stakeholders as the key success factor . Furthermore, the empirical research results indicate that software architects not only interact with stakeholders, but are also forced to help them identify architecturally significant requirements. The reason for this is that for the most part stakeholders do not know what they want, or, if they do know what they want, they are unable to grasp the implication of the requested requirements. What is crucial here is the role of the architect, who must recognise drastic changes in the approach and the architecture itself based on requirements, and tell the client what is feasible and what is not (RESP.10). In other words, he/she is responsible for presenting stakeholders with the value that a particular architectural solution provides to the user, as well as with entailed costs and risks (RESP.16). The responsibility for the choice of an architectural solution is rests mostly with the client (RESP.3), but in case the client insists on developing a risky solution, the software architect must make sure that he/she can always easily switch to the one that suits him/her better (RESP.1).

Empirical results show that, in an organisation with a large number of teams, the responsibility for the development of an architecture is diversified into several different architects' roles. In addition to the software architect, who works with teams on particular details of software architecture, there is also the role of the system/enterprise architect, who is responsible for the setup of the architecture of the entire product. The solution architect plays a role of an architect responsible for the implementation of an architectural solution in the target organisation (RESP.17). Unlike Babar (2009), who links the role of the solution architect to the responsibilities focussed on managerial aspects, in practice, respondents allocate these responsibilities to the system architect, whereas the role of the solution architect is linked by respondents to the team delivering a software solution. Babar (2009)identifies the role of the implementation architect as the one referring to his/her responsibility to monitor the implementation of user narratives, provide technical mentorship to programmers and monitor whether there are negative effects of refactoring. The respondents link these responsibilities to the role of the software architect.

The research results lead to a conclusion that the owners of an architecture (the system architect, the software architect, the solution architect etc.) on a complex project must have top-notch technical knowledge and the solid knowledge of the technical domain. The responsibilities of the architecture owner are as follows:

- to identify initial architecturally significant requirements at the beginning of the project, together with the client;
- to set up (envision) the initial architectural solution for the main part of the software;
- to review the architecture continuously in terms of meeting non-functional system requirements (through the continuous integration of the code, a set of metrics and tests);
- to have good collaboration with team members, aimed at sharing ideas and resolving problems;
- to lead technical discussions with other team members;
- to instruct other team members and provide mentorship in resolving architectural problems during the implementation phase;
- to understand the existing infrastructure, standards and technical solutions of target organisations;
- to know the possibilities and limitations of as many technologies as possible;
- to follow trends in terms of possible options of an architectural solution;
- to provide the visibility of architectural requirements on the product backlog and backlog;
- to manage the technical debt from the beginning of the project;
- to make timely decisions.

Kruchten (2009) identified a set of characteristics that the software architect ought to have: he/she should be a good visionary (so as to view and set up the picture of the system globally), able to make decisions, a skilful communicator, capable of resolving architectural problems that programmers cannot resolve. In addition to this, the respondents also pointed out the significance of the architect's leadership skills in the team (RESP.8) as well as the power of abstraction (RESP.5).

Conclusion and research limitations

The formal role of the software architect is unusual for agile development processes. Agile teams are cross-functional and all team members share the responsibility for an architecture. The results of the conducted empirical research, however, present a different picture. Actually, the software architect in all agile teams included in the research plays a formal role, which, however, is significantly different from the traditional one. The difference is, first of all, in the continuous involvement of the architect throughout the development process.

The empirical results show that the other team members also make a contribution on their own part in the development of software architecture by continuously communicating and closely collaborating with the software architect. The members of the agile team participate in discussions about the initial setup of the architecture, but the final decision on the higher level architecture design is up to the architects and the client. A greater responsibility of the team members is noticed in the phase of the detail design, where they are expected to independently work on a solution, with the possible mentorship of the architects in the case of problems that they themselves are unable to independently resolve. This leads to a conclusion that the detail design is the part where the software architect's role of a coordinator is recognised. As the project commissioner and the budget owner, the client very often makes the final decision when choosing an architectural solution.

In addition to the software architect, in practice, there are two more formal roles responsible for an architecture played by agile teams: the roles of the system architect and the solution architect. In most cases, the system architect also has a managerial position in the organisation and is responsible for the architecture of the whole system. In other words, the system architect takes care of the coordination of the software architect's architecture with the architecture of the entire system. Software architects are members of development teams and, for the most part, are senior programmers and team leaders, too. Solution architects are parts of the software delivery team and therefore cooperate closely with employees who carry out operative activities in the target organisation.

In practice, agile teams are aware of the fact that responsibility for the architecture should rest on the whole team rather than on architects only. However, such a method of work requires experienced individuals in the team, with top technical knowledge, which is the key challenge faced by agile teams in practice. The rapid development of the IT industry in Serbia, on the one hand, and the limited workforce market, on the other, have brought about the fact that agile teams are to a great extent comprised of inexperienced individuals. The respondents see another problem in the inadequate higher education system in Serbia. College and university curricula do not correspond to the market needs, or more specifically, there is an inadequate study of current technologies, program languages, frameworks and architectural patterns. Also, the respondents opine that universities devote an insufficient curriculum time to knowledge and skills from the domain of software architecture and agile process development. Another problem they pointed out is that not a single higher education institution in Serbia profiles students for the profession of a software architect.

The results of the systematic literature review of lead to a conclusion that no similar empirical research in the area of software architecture in agile processes has been conducted in Serbia so far. This fact is also the limitation of the completed empirical research, as the obtained results cannot be compared with the results obtained by other researchers in the territory of Serbia. SM

References

Abrahamsson, P., Babar, M. A., & Kruchten, P. (2010). Agility and Architecture: Can They Coexist? *IEEE Software, 27* (2), 16-22.

Ambler, S. W., & Lines, M. (2013). *Disciplined agile delivery (1st ed.)*. Boston, MA: IBM Press.

Babar, M. A. (2009). An exploratory study of architectural practices and challenges in using agile software development approaches. In *Joint working IEEE/IFIP conference on software architecture 2009 and european conference on software architecture (WICSA/ECSA)* (pp. 81-90). Cambridge, UK: IEEE.

Babar, M. A. (2014). Making Software Architecture and Agile Approaches Work Together. In M. A. Babar, A. W. Brown, & I. Mistrik (Eds.), *Agile software architecture* (1st ed., pp. 43-76). Waltham, MA: Elsevier.

Babar, M. A., Ihme, T., & Pikkarainen, M. (2009). An Industrial Case of Exploiting Product Line Architectures in Agile Software Development. In *13th international conference on software product lines (SPLC)* (pp. 171-179). Pittsburgh: CMU.

Babar, M., & Abrahamsson, P. (2008). 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). Limerick.

Beck, K. (2004). *Extreme Programming Explained: Embrace Change (2nd ed.)*. Boston, MA: Addison-Wesley.

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). New York, NY: IEE.

Blair, S., Watt, R., & Cull, T. (2010). Responsibility-Driven Architecture. *IEEE Software, 27 (2)*, 26-32.

Boehm, B. (2002). Get ready for agile methods, with care. *IEEE Computer, 35* (1), 64-69.

Boehm, B., & Taylor, R. (2005). Challenges to implementing agile processes in traditional development organizations. *IEEE Software, 22* (1), 30-39.

Buschmann, F. (2012). A week in the life of an architect. *IEEE Software, 29* (3), 94-96.

Coplien, J. O., & Bjornvig, G. (2010). *Lean architecture for agile software development*. UK: John Wiley and Sons.

Dyba, T., & Dingsoyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology, 50*, 833-859.

Faber, R. (2010). Architects as service providers. *IEEE Software, 27* (1), 33-40.

Hadar, I., & Sherman, S. (2012). Agile vs. Plan-Driven Perceptions of Software Architecture. In Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop (pp. 50-55). Zurich, Switzerland: IEEE.

Helmer, O., & Rescher, N. (1959). On the Epistemology of the Inexact Sciences. *Management Science, 6* (1), 25-52.

Hopkins, R., & Harcombe, S. (2014). Agile Architecting: Enabling the Delivery of Complex Agile Systems Development Projects. In I. Babar, M. I., Brown, A. W.,Mistrik (Eds.), *Agile Software Architecture: Aligning Agile Processes and Software Architectures* (1st ed., pp. 291-314). New York, NY: Elsevier.

Ihme, T., & Abrahamsson, P. (2005). 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).

Keeney, S., Hasson, F., & McKenna, H. (2011). *The Delphi Technique in Nursing and Health Research (1st ed.)*. London: Wiley-Blackwell.

Kitchenham, B. (2004). Procedures for performing systematic reviews. Retrieved January 20, 2015 from Departamento de Informática e Estatística: http://www.inf.ufsc.br/~awangenh/kitchenham.pdf

Kruchten, P. (2007). Voyage in the agile memeplex. *ACM Queue, 5* (5), 38-44.

Kruchten, P. (2008). Situated agility: context does matter, a lot. In *9th International conference on agile processes and eXtreme programming in software engineering*. Limerick.

Kruchten, P. (2009). Agility and architecture: an oxymoron? In *SAC 21 Workshop: Software Architecture Challenges in the 21st Century*.

Kruchten, P. (2013). Contextualizing agile software development. *Journal of Software: Evolution and Process, 25* (4), 351-361.

Lycett, M., Macredie, R. D., Patel, C., & Paul, R. J. (2003). Migrating agile methods to standardized development practice. *IEEE Computer, 36* (1), 79-86.

Madison, J. (2010). Agile architecture interactions. *IEEE Software, 27* (1), 41-48.

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

Miles, M.B., Huberman, A. . (1994). *Qualitative data analysis: An expanded sourcebook (2nd ed.)*. Sage.

Nord, R. L., & Tomayko, J. E. (2006). Software architecture-centric methods and agile development. *IEEE Software, 23* (2), 47-53.

Parsons, R. (2008). Architecture and agile methodologies—how to get along. In *WICSA*.

Stal, M. (2014). Refactoring Software Architectures. In *Agile Software Architecture: Aligning Agile Processes and Software Architectures* (pp. 130-152). W: Elsevier.

Thapparambil, P. (2005). Agile architecture: pattern or oxymoron? *Agile Times, 6* (1), 43-48.

✉ **Correspondence**

Mirjana Marić

Faculty of Economics in Subotica
Segedinski put 9-11, 24000, Subotica, Serbia

E-mail: maricm@ef.uns.ac.rs