Documenting Agile Architecture: Practices and Recommendations

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 foundation of every software product, regardless of the process used for its development. Traditional architecture development based on three architectural phases – architectural analysis, synthesis and evaluation is considered highly ceremonial, due to the great number of artifacts it produces. In agile development, architecture is generated gradually with each iteration, as a result of continuous code refactoring, not some predefined structure. In other words, agile processes do not include any of the traditional phases (analysis, synthesis, and evaluation) of the architecture development process, while self-documenting code is the predominant form agile architecture documentation.

Excessive documentation is considered wasteful in agile development processes. However, complete elimination of documentation results in "evaporation" of architectural information and knowledge, which may compromise the entire development process. Therefore, development of complex software systems requires an architecture documenting strategy positioned between the described extremes.

This paper presents results of theoretical and empirical research on documenting software architecture in agile development processes. Subsequent to the systematic literature review, an empirical research based on the classic Delphi method was carried out on a sample of 20 expert practitioners. In addition to an overview of current architecture documenting practices, the paper proposes structures of two artifacts for documenting agile architecture of complex systems, developed with regard to the results of the empirical research. These artifacts contain short descriptions of architectural decisions and rationale behind them.

Keywords: Agile software development · Software architecture · Architecture documentation

1 Introduction

Documentation is definitely among the most important explicit architectural practices. While overemphasized in traditional development, in agile development it was almost entirely replaced by the practice "source code is the ultimate documentation", in line with the agile principle of "working software over comprehensive documentation".

S. Wrycza (Ed.): SIGSAND/PLAIS 2016, LNBIP 264, pp. 56–71, 2016.
DOI: 10.1007/978-3-319-46642-2_4

Documenting architecture and architectural decisions is among the greatest challenges of complex system development using agile processes. In other words, architecture has a significant role in development of complex business software solutions, and therefore, quality self-documenting code cannot be the only documenting practice; it is necessary to incorporate some of the traditional documenting practices, to an appropriate extent. In agile processes, excessive documentation is considered a waste, which may impair the agility of the development process and compromise early delivery of value to users. On the other hand, complete elimination of this architectural activity results in "evaporation" of architectural information and knowledge, poor comprehension of the architecture, and impaired communication within the development team, which can cause chaos and failure of the development process [9–11].

In line with the arguments made on this issue in the existing literature, we believe that the most important step in overcoming this prevailing problem is to identify explicit architectural activities that could be incorporated into agile development processes, as to ensure that adequate amount of architecture documentation is generated. Hence, the research goal was define the structure of essential architecture documentation in agile processes, based on a set of empirically identified key architectural activities.

The essential purpose of architecture documentation is to help the development team in understanding how the future software solution is organized, how it is supposed to work, and what are the reasons being key architectural decisions made throughout the development process. This provides the development team with a clearly defined architectural strategy, which ensures consistency and reduces the probability of taking wrong turns in the development.

Architectural artifacts must remain visible and transparent throughout the whole development process. This requires explicit statement and inclusion of architectural requirements into the Product Backlog agile artifact. Addition of architecturally significant requirements to the Product Backlog, as an explicit architectural activity, involves compiling a comprehensive list of product requirements – functional, non-functional, and requirements for future system changes, as well as ranking them by significance and required implementation time. Documents shuold not be long and complex, but rather concise and understandable, without burdening details, as to be effortlessly understood, revised, and used for communication purposes.

With regard to the topic and research problem previously described, the following research questions were formulated.

RQ1. What are the most commonly used architecture documenting practices in agile development processes?

The answer to this research question is as result of a state of the art exploration, by means of systematic literature review. Results of the literature review are presented in Sect. 3 – Theoretical Background: Systematic Literature Review.

RQ2. Which explicit architectural activities are significant to documenting architecture in agile development processes?

The answer to this question was obtained thru elaboration on results of qualitative and quantitative components of the conducted empirical research, and served as a basis

for the proposed structure of key architecture artifacts for complex systems development using agile processes.

2 Research Methodology

Research design, depicted in Fig. 1, was developed by modifying the framework provided by [1]. Sequence of research activities, along with the techniques used, are presented in Fig. 1. Research problems and research questions described in the introduction represent results of the r*esearch subject identification* phase.

Activity *state of the art exploration* within *theoretical research execution* phase provided the answer to RQ1. The activity was carried out by means of a systematic literature review, in accordance with recommendations by Kitchenham [2]: planning the review, conducting the review, reporting the review. The stages associated with *planning the review* include: (1) identification of the need for a review, (2) development of a review protocol. Stages associated with *conducting the review* are: (1) identification of research, and (2) selection of primary studies, (3) study quality assessment, (4) data extraction & monitoring, and (5) data synthesis. *Reporting the review* is a single stage phase, further described in the Theoretical Background section.

An overview of the total number of hits per each electronic database is given in Table 1. Analysis of search results resulted in 10 relevant papers concerning issues of architecture documentation in agile development processes.

Table 1. Number of hits per each database

Source	Number of hits with the keywords: agile software architecture/agile methods and architecture/agility and architecture/agile documenting	Number of papers selected for further analysis
IEEE Xplore	701	43
ScienceDirect	46	12
ACM Digital library	237	12
Total	984	67

Empirical research realized within the empirical research execution stage involved applying the classic variation of the Delphi method in three iterations [4, 5]. Nature of the research problem necessitated purposive selection of sample units ($n \geq 20$). Hence, the sample was composed of experts experienced in agile development and software architecture design. Results obtained in the empirical research execution phase provided the answers to RQ2.

The first iteration of the empirical research represented its qualitative component. Data collection was carried out by means of a semi-structured interview. The interview consisted of a set of predefined questions, previously evaluated by experts. Interviews were conducted face-to-face and recorded, as to ensure greater accuracy and consistency

of collected data. Transcribed interview data was subject to qualitative analysis in NVivo suite (using thematic content analysis technique), in line with recommendations by Miles i Huberman [3]. Results of the first iteration were obtained within the state of the practice exploration activity.

The second and the third iteration of the empirical research constituted its quantitative component. A questionnaire with checklists and Likert-type assessment scales served as the research instrument in both iterations. Gathered data was subject to qualitative analysis in SPSS, using following methods: descriptive statistics, Efron's bootstrapping, Cohen's kappa coefficient, Chaffin-Talley index of individual stability, McNemar's test, and McNemar-Bowker's test.

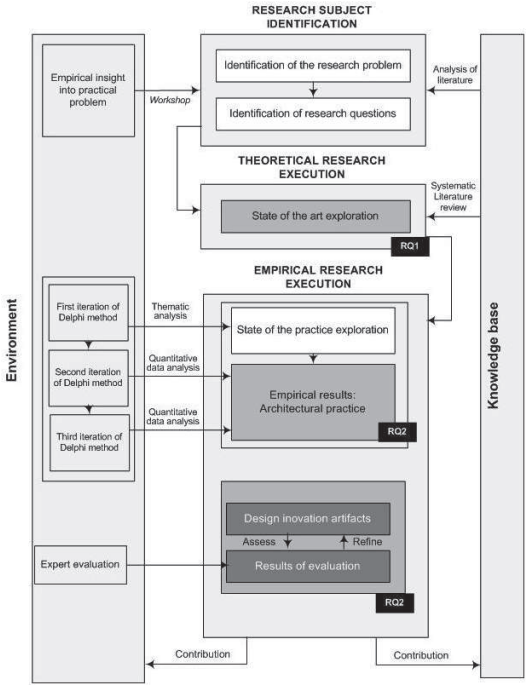

Fig. 1. Research design

3 Theoretical Background: Systematic Literature Review

Results of the literature review indicated that a significant portion of architectural problems originates from an essential conflict between the requirement of minimalism in agile methods and the need for well-documented architecture in complex systems [6]. The need for adequate management of architectural knowledge in agile processes is further amplified by increasingly present global software development [7].

According to the reviewed literature, most cases in the industry belong to one of the extremes: overly extensive documentation or absence of architecture documentation [8].

Poor documentation causes “evaporation” of architectural information and knowledge [9], inadequate architectural understanding, and impaired communication within the development team, which can cause chaos failure of the development process [10]. On the other hand, overly excessive documentation is considered a waste of time, resources, and distancing from the essence [10].

Excessive documentation impairs efficiency of agile processes, since stakeholders, such as programmers, users, and clients lack time and energy to study traditional architectural artifacts in order to understand the architecture. Programmers want clear and decisive design guidelines, while users and clients want to be sure that the architecture supports their business needs. Architects who resume someone else’s work want clearly emphasized architecture’s key aspects, including reasons for architectural decisions considered and implemented by the previous architect [11].

Faber [12] believes that it is the software architect, with the role of a service provider, who is responsible for maintaining a central position between inadequate and excessive documentation of development guidelines.

Several researchers have recognized the importance of establishing balance between excessive (traditional) and insufficient (agile) architecture documentation, to enhance preservation and dissemination of architectural knowledge, and suggested various approaches for resolving this architectural problem.

Tyree and Akerman [11] see the solution primarily in documenting architectural decisions and rationale behind them. They consider adequate documentation of architectural decisions to be an efficient tool for communication with stakeholders, which enables them to understand the architecture quickly and effortlessly. In addition to that, it is efficient in guiding system implementation, as well as monitoring compliance of technical implementation with the original requirements. Documented decisions can also be used as means for evaluating architectural solutions [11].

Hadar, Sherman, Hadar and Harrison [8] proposed a template for documenting software architecture that is consistent with the agile philosophy and lean documentation. The proposed architectural document originated form an empirical research. It is focused solely on the most relevant information on the architectural solution designed for a particular release. It consists of four principal sections: product overview, product goals for upcoming release, product architecture overview as a whole, and non-functional requirements. Architecture is described at four conceptual levels: system components layer, common and cross concern components layer, external integration components layer, and functional components layer. The approach relies on a tool for automatic documentation generation. The documenting tool is connected with the architecture modeling tool by a common database, so that changes in the model are updated in the documentation [8].

Pareto, Sandberg, Eriksson and Ehnebom [10] proposed a method for prioritizing architectural documentation for projects within large organizations that involve corrective actions over existing architecture. According to them, architecture documentation designates a set of models, viewpoints, and views covering different architectural aspects from various stakeholders’ perspectives. The proposed method combines collaborative and analytical techniques involving different stakeholders, aimed at identifying architectural areas that require improvement.

Eloranta and Koskimies [13] see inadequate documentation in agile processes as the obstacle to distribution of architectural knowledge. They point out that, in projects of higher scale and complexity, communication between developers and stakeholders is not sufficient, and propose the concept of Architecture Knowledge Management to be integrated into the Scrum process. The approach involves development of an architectural knowledge base and utilization of a method for evaluating architectural decisions. The proposed method (DCAR) analyses each architectural decision the moment it is made, from bottom to top. Due to its incremental nature, which enables piecewise evaluation of architecture, DCAR method is suitable for agile processes. After analysis, architectural decisions and their justifications are recorded in the architectural knowledge base, which is a sort of an information system. Architectural database can automatically generate (on-line or printed) architectural reports or documents for specific purposes, using stored information [13].

In his research on documentation issues in agile processes, Babar [14] highlighted the use of a modified traditional documentation practice in the form of a Software Architectural Overall Plan, however, limited only to conceptual description of architecture. Remaining design decisions are described in Wikis. Responsibility for analysis and selection of an architectural solution is shifted to the client. Architectural evaluation carried out by the Architecture Review Board, where programmers evaluate proposed solutions, is rather informal [14].

However, Falessi et al. [15] have shown that agile programmers have positive attitudes towards software architecture, and that they perceive architectural artifacts as beneficial to communication between developers, later design decisions, documentation and product evaluation [15].

4 Results of the Empirical Research

Quantitative results suggest that agile teams recognize the importance of architectural documenting, since the proportion of respondents who rated this explicit architectural practice as significant was 0.85.

Analysis of qualitative data related to one of the identified categories – architectural documenting – lead to a conclusion that, in practice, agile teams develop a set of informal software architecture documents, covering some basic development guidelines, such as code standards, error handling, etc.

As opposed to traditional documenting, which involves making formal artifact with views describing different perspectives of the architecture (e.g. RUP 4 + 1 view), agile teams tend to use Wikis, without a predefined form.

Architectural decisions are most commonly made and clarified to team members in front of the whiteboard. Hence, the most common manner of documenting architectural decisions and rationale behind them are informal diagrams drawn on a whiteboard, which are later photographed and stored in a Wiki, or distributed to system stakeholders via email. Documents in the form of images are also used for communication and collaboration with stakeholders.

Most of the respondents emphasized use of tools that ease documenting activities. Most frequently used tools include Jira, GitHub, Word and Excel, which are also used for collaboration with stakeholders.

Development of formal models and use of specialized tools, such as Rational Rose or Enterprise Architect are typical to traditional development. Unlike in the traditional development, where implementation is preceded by comprehensive diagramming, agile teams dismissed this as a waste of time and resources, stating that development and maintenance of such diagrams is not possible in a dynamic, volatile environment.

Basic UML diagrams are made only for communication within the development team, when there is no better way to present a particular solution. However, UML is avoided in communication with external stakeholders, since software development practitioners feel that business people have difficulties understanding it.

Architectural documentation contains a description of the problem being solved, descriptions of architecturally significant functional and non-functional requirements, as well as the decision on the technology stack. Some of the non-functional requirements are extracted from descriptive elements, i.e., what clients expect from the system throughout its use; they are subsequently specified as non-functional requirements and their validity is checked with the client in the verification process.

The documentation also contains the initial architectural solution, up to an intermediate level of abstraction. Defining classes, their attributes and methods is not a common practice, unless the system is exceptionally complex, and team members lack skills and experience in design. Respondents also noted that architecture documentation is very helpful when new team members join the development team.

Architectural documents most commonly contain some initial deployment model and an initial database model. Information obtained thru analysis of the user story being implemented is also recorded during development. Most commonly, this involves examples of positive and negative tests for that user story, impact on security aspects of the architecture, and similar.

In several cases, the documenting process begins with the development of acceptance tests in a Word document, and ends with integration tests, which represent the final specification of the product.

Several respondents highlighted that it is useful, even desirable when architectural documentation is formally revised and verified by a team of experts, and that the implementation of the solution should begin only after the architecture had been approved. However, this is rarely the case in practice, as is remarked by one of the respondents: "… Once software architect completes that document, it is not good practice to upload it to an internal site and be done with it; it is desirable to send it to other software architects, head of QA and head of security for their verification. If they say that 'this is it', it is finally sent to chief architect for approval. Once they read the whole document and approve it, than it is set for implementation. Implementation of user stories should not begin prior to the approval; however, this is not the case, since the chief architect lacks time to read the document, so every document ended up pending approval."

Qualitative analysis of respondents' answers revealed that explicit architectural activity of documenting detailed design should not be incorporated into agile development processes. The proportion of respondents who classified this practice as significant was 0.1. In other words, 90 % of surveyed experts classified documenting detailed design as insignificant. Such high level of agreement among experts indicates that practitioners find documenting this activity unnecessary in agile processes. They stated several reasons, including that it requires additional time and effort, as well as that programmers already possess skills necessary to independently, or with technical guidance from a software architect, develop the detailed design.

Results indicate that there nearly all agile practitioners agree on the importance of the explicit architectural practice of Product Backlog prioritization. Numerous respondents noted that the architecture owner's participation in Product Backlog prioritization is just as important as the product owner's. Product owner, as a representative of the user, sets priorities from the point of value delivered to the user, while architecture owners need to prioritize technical features and the order of their implementation over iterations, based on previous technical dependency analyses and risk assessments. Architecture owner's technical suggestion depends on availability of team members, tools, and so on. The architecture owner should rank architectural requirements in accordance with the following factors.

- Value delivered to stakeholders with the implementation of a particular architectural requirement: critical/important/useful.
- Requirement's impact on architecture: none/extends/alters. Some functional requirements can be classified as critical (high priority from the point of value to stakeholders), and yet have no impact on architecture, and vice versa.
- Risks to be mitigated (performance, availability, component suitability)
- Other tactical goals and limitations.

The Product Backlog must be updated throughout the development process, after each iteration and after each release.

Creation and maintenance of a Product Backlog can be done efficiently using agile project management tools, which usually come bundled with templates. For that reason, this artifact was not particularized in this paper, that is, its structure is not proposed in the following section. We only point out that this agile artifact needs to encompass all kinds of requirements: functional, non-functional and requirements for future system changes.

All architectural activities, identified as a result of the empirical research, that were classified as significant to the development of software architecture in agile development processes are given in Table 2. This comprehensive list of significant practices served as a basis for developing structures of two key architectural artifacts – the Vision Document and the Software Architecture Document, which will be presented in the following chapter.

Table 2. Empirically identified explicit architectural practices significant to development of complex software solutions using agile processes

Empirically identified explicit architectural practices	Proportion of respondents who classified the practice as significant
Functionality interdependence analysis	1
Forming a suitable team and choosing a software architect with regard to the problem being solved	0.95
Understanding of the business problem	0.95
Code review	0.95
Development of coding rules and other guidelines for system design	0.95
Reviewing present architectural solutions	0.95
Active discussions with stakeholders aimed at analyzing and understanding the business	0.9
Identification of architecturally significant requirements	0.9
Risk analysis aimed at identifying and isolating complex areas	0.9
Examining technology suitable for implementation	0.9
Identification and definition of basic structures (modules) for the system core, as well as their relations	0.9
Testing system performance and other critical non-functional requirements	0.9
Identifying common components and common infrastructure of the functionality set of the future release	0.9
Use of Use Case technique	0.9
Managing dependencies with external systems that the system interacts with throughout a release	0.9
Static code analysis	0.9
Test Driven Development with focus on non-functional requirements	0.9
Explicit identification of non-functional requirements	0.9
Prioritizing Product Backlog, with regard to business value, architectural risk, and architectural impact	0.9
Project scoping	0.85
Analysis of dependencies between functional requirements and architectural elements during release planning	0.85
Configuration management	0.85
Tracking architectural task in the Backlog or a Kanban board throughout the whole development process	0.85

(*Continued*)

Table 2. (*Continued*)

Empirically identified explicit architectural practices	Proportion of respondents who classified the practice as significant
Generating top level documentation	0.85
Architecturally significant requirements are identified as a result of consultations with stakeholders, analysis of functional and non-functional requirements, and anticipation of future business goals	0.85
Creation of a Product Backlog, with all functional, non-functional, and requirements for future system change	0.85
Architectural planning beyond one release	0.85
Evaluation of present architecture before a new release	0.85
Continuous review of architecture	0.85
Prioritization of the Product Backlog	0.8
Defining the basic data architecture	0.8
Development of a deployment model	0.8
Validation of critical architectural requirements and design concepts using prototypes	0.8
Specification of integration tests	0.8
Estimation of technical user stories and time required for their development	0.8
Selection of a suitable implementation framework	0.8
Assessment of the existing infrastructure in the target organization	0.8
Identification of architectural elements that need to be more flexible	0.8
Architectural spikes	0.8
Prototyping	0.8
Evaluation of architecture by the architectural board	0.85
Scenario analysis of stakeholders' interaction with the system, with focus on non-functional requirements	0.85
Identification of key stakeholders	0.75
Formal architecture review	0.75
Regression testing	0.75
Load testing future architecture	0.75
Specification of test cases	0.75
Examining third-party libraries	0.75
Identification of future goals and business development	0.75

(*Continued*)

Table 2. (*Continued*)

Empirically identified explicit architectural practices	Proportion of respondents who classified the practice as significant
Release planning that includes a strategy for examining legacy systems, dependencies with partner or other third-party products, and backward compatibility of data	0.7
Specification of acceptance tests	0.7
Development of QA tests	0.7
Time boxed proof of concept	0.7
Market research and domain analysis aimed at better identification of architectural requirements	0.7
Analysis of dependencies between functional and non-functional requirements	0.7
Identification of necessary changes to the architecture, prior to the next release	0.7
Explicit identification of requirements related to geographic dispersion of the future system	0.7

4.1 Structure of Artifacts for Documenting Software Architecture of Complex Business Systems in Agile Processes

The Vision Document, states the business problem that the future system should solve, as well as the set of business requirements essential for identifying architecturally significant requirements. In addition to that, it includes an overview of the target organization's existing infrastructure, and other restrictions that also influence the choice of an architectural solution. The document consists of 11 main sections: Problem Being Solved, System Stakeholders, Target Organization's Business and Information Architecture, Organization's Goals and Development Paths, Features of the Future System, Target Organization's Software and Technical Infrastructure, List of Architectural Risks, Team, Architecturally Significant Requirements, Technological Issues, Choice of the Architectural Solution.

The Software Architecture Document contains information on the design of the architectural solution. The intention was that it should contain only the most relevant architectural decisions and be as comprehensible as possible. Such straightforward overview should simplify communication between the software architect and all stakeholders, particularly the development team, which can use it to effectively guide the implementation of the system. The Document is comprised of 8 main sections: Modules of the Main Part of the System and Their Dependencies, Defining Subsystems and Layers for the Identified Modules, Defining Key Elements, Interactions and Design Mechanisms, Architectural Decisions on the Data Model, Identification of Architectural Elements That Need to Be More Flexible, Architectural Decisions on Deployment Model, and Architectural Decisions on the Implementation Model. The listed sections contain rationale of architectural decisions and graphical representation based on

informal models, drawn on paper or a whiteboard, and captured as photographs. With such a concise document, reviewers can be more effective, that is, quickly provide precise comments and recommendations. The section "Decisions on Changes in the Architecture and Rationale behind Them" allows for effortless tracking of changes made in the architecture in different releases.

The structure of the documents is given below.

A. Vision Document
Problem Being Solved

Problem statement	
Key benefits of a successful solution to the problem	

System Stakeholders

Stakeholder title	Key responsibilities	Present expectations/expectations from the future system

Target Organization's Business and Information Architecture
Organizations workflow presented with an informal flowchart.
Organization's Goals and Development Paths
A short description that will serve as a basis for compiling a list of requirements for future system changes.

Features of the Future System

Business need	System feature (general level)	Reason for implementation of the feature (value for customer)

Target Organization's Software and Technical Infrastructure

- Software application currently in use within by the organization (brief description);
- How the present software applications are used (brief description);
- Interdependence of present software applications;
- Existing relevant hardware infrastructure;

After the analysis and documenting of software and technical infrastructure, it is necessary to answer the following questions:

– Is the technology already implemented within the target organizations suitable for the new system?
– Would the implementation of new technology increase risk?
– Does development success depend on novel technologies that have not yet been fully tested?
– Is reuse of software components that exist in the target organization justified?
– Are existing software components developed or purchased?
– Do dependencies on other systems include ones outside the organization?
– Do interfaces for their communication already exist, or should they be developed with the new system?

List of Architectural Risks

Risk	Architecture's exposure to risk	Risk scope	Mitigation strategy

Systems scoping involves updating the list of risks in the table, by answering the following set of questions:

– Is the amount of transactions the system is expected to support reasonable?
– Is the amount of data the system should operate with reasonable?
– Are there any unusually technically challenging requirements that the development team has not encountered so far?
– Are the identified requirements relatively stable and clear?
– Are there any extremely inflexible non-functional requirements (e.g. that the system must never fail), are requirements complex?
– Are domain experts available?
– Does the team have enough people, with adequate skills and experience?

Risks identified during verification and implementation of the architecture

Team

Name	Experience, skills, expertise	Assigned position/responsibilities

Architecturally Significant Requirements:

– Key non-functional requirements
– Non-functional requirements
– Requirements for future changes
– Requirements associated with geographic distribution
– Real-world limitations

Technological Issues

- Decision on the technology stack
- Results of option trend analysis
- Selected implementation framework
- Selected programming language
- Decision on use of existing class libraries
- Decision on reuse of existing software components

Choice of the Architectural Solution

- Defined criteria for selecting an architectural solution
- Selection of a technique for evaluating potential architectural solutions
- Results of analysis and evaluation of potential architectural solutions (strengths/weaknesses)
- Justification of the choice of architectural solution

B. Software Architecture Document

Modules of the main part of the system and their dependencies

Graphic representation based on an informal model, drawn on paper or a whiteboard and captured as a photograph. General overview of the architecture illustrates the essence of the proposed architectural solution and main building blocks (elements, modules) that will constitute the architecture of the system.

Defining subsystems and layers for the identified modules

Graphic representation based on an informal model, drawn on paper or a whiteboard and captured as a photograph. Organization of subsystems for each identified module should represent the system design at the highest level of abstraction.

Defining key elements, interactions and design mechanisms

- Graphic representation based on an informal model, drawn on paper or a whiteboard and captured as a photograph. The representation contains initial design elements that will be modified and detailed during implementation, as well as interactions between key abstractions, which specifies a method of communication between design elements.
- This section should also include a list of identified design mechanisms.
- Decision on the method of their implementation is documented in the implementation phase.

Architectural decisions on the data model

- Decision on the type of database to be developed.
- Problems and solutions related to data compatibility.
- Initial setup of a data model, which will be extended and detailed during implementation

Identification of architectural elements that need to be more flexible

Decision on architectural elements that need to be more flexible, due to the nature of identified requirements for future system changes.

Architectural decisions on the deployment model

- Decision on deployment model solution
- Initial conception of the deployment model

Key decisions on the implementation model

Key decisions on the implementation model, without many details.

Decisions on changes in the architecture and rationale behind them

Includes decisions that resulted from verification and review of the system architecture throughout the implementation.

5 Conclusion

Results presented in this paper are a part of the research conducted for the development of the doctoral dissertation "Methodological Framework for Developing the Software Architecture of Business Software in Agile Processes". The results indicate that agile practitioners do not renounce architecture documenting activities, but rather consider them significant in the development of complex systems.

Based on the results of this research, the authors recommend that the structure of key artefacts for documenting architecture be formally defined. For this reason, this paper proposes the structure of two key artefacts, which can be tailored to specifics of the projects. The basic idea is that documents should not be too extensive, and should only cover the most important architectural decisions and rationale behind them. Architectural decisions would still be made in an agile manner, using informal models and means for representation both the team and the stakeholders are already accustomed to.

The proposed artifacts facilitate preservation of architectural knowledge, improve team collaboration, and assist in understanding the business problem and solutions to it, which altogether should reduce the possibility of taking wrong turns in the development. In addition, the proposed artifacts do not obstruct the agility of the development process, since they do not rely on traditional documenting practices, which involve use of formal languages for modeling software architecture.

References

1. Hevner, S., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Q. **28**(1), 75–105 (2004)
2. Kitchenham, B.: Procedures for performing systematic reviews (2004)
3. Miles, M.B., Huberman, A.: Qualitative Data Analysis: An Expanded Sourcebook, 2nd edn. Sage, Thousand Oaks (1994)
4. Helmer, O., Rescher, N.: On the epistemology of the inexact sciences. Manage. Sci. **6**(1), 25–52 (1959)
5. Keeney, S., Hasson, F., McKenna, H.: The Delphi Technique in Nursing and Health Research, 1st edn. Wiley, London (2011)

6. Hadar, I., Sherman, S.: 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 (2012)
7. Clerc, H., Lago, V., Vliet, P.: Architectural knowledge management practices in agile global software development. In: Sixth IEEE International Conference On Global Software Engineering Workshops, pp. 1–8 (2011)
8. Hadar, J.J., Sherman, I., Hadar, S., Harrison, E.: Less is more: architecture documentation for agile development. In: 2013 6th International Workshop Cooperative and Human Aspects of Software Engineering (CHASE), pp. 121–124 (2013)
9. Hadar, G.M., Silberman, E.: Agile architecture methodology: long term strategy interleaved with short term tactics. In: Companion to the 23rd ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications, OOPSLA Companion 2008, pp. 641–651 (2008)
10. Pareto, S., Sandberg, L., Eriksson, A., Ehnebom, P.: Prioritizing architectural concerns. In: 2011 Ninth Working IEEE/IFIP Conference on Software Architecture (WICSA), pp. 22–31 (2011)
11. Tyree, A., Akerman, J.: Architecture decisions: demystifying architecture. IEEE Softw. **22** (2), 19–27 (2005)
12. Faber, R.: Architects as service providers. IEEE Softw. **27**(1), 33–40 (2010)
13. Eloranta, K., Koskimies, V.P.: Aligning architecture knowledge management with scrum. In: Proceedings of the WICSA/ECSA 2012, pp. 112–115 (2012)
14. Babar, M.A.: An exploratory study of architectural practices and challenges in using agile software development approaches. In: 2009 European Conference on Software Architecture. Joint Working IEEE/IFIP Conference, pp. 81–90 (2009)
15. Falessi, D., Cantone, G., Sarcia', S.A., Calavaro, G., Subiaco, P., D'Amore, C.: Peaceful coexistence: agile developer perspectives on software architecture. IEEE Softw. **27**, 23 (2010)