Computer Science and Information Systems 15(1):79–109 https://doi.org/10.2298/CSIS160820038M

Traditionalisation of Agile Processes: Architectural Aspects

Predrag Matkovic, Mirjana Maric, Pere Tumbas, and Marton Sakal

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

Abstract. Mechanisms of agile processes, suited for cost reduction and timely reaction to dynamic market changes, have also been recognized as useful in the development of complex software solutions. Recent studies focused on expansion of agile processes point to a viable possibility for coexistence and integration of complementary elements of agile and traditional development. Within the scope of this paper, this phenomenon is referred to as traditionalisation of agile processes. Software architecture modeling is one of the most sensitive issues associated with incorporation of elements of traditional development into agile processes. The goal of this paper was to determine how suitable particular explicit architectural practices are for incorporation into agile development processes. A mixed method research was carried out for this purpose. Qualitative component of the research resulted in identification of explicit architectural practices suitable for application in agile development processes. Their significances were determined by means of the quantitative component, realized in the form of an empirical research. The research confirmed that emergent architecture in agile processes is not sufficient for the development of complex software solutions, and that agile processes need to incorporate certain explicit architecture practices. Research results revealed that the agile community has an affirmative attitude towards the idea of incorporating explicit architectural practices into agile development processes, with overall agreement on the significances of particular explicit architectural practices for the development of architecture of complex software systems.

Keywords: software process models, agile process, software architecture modeling, scaling up agile processes.

1. Introduction

Agile and lean practices originate from the post-World War II period, when Japanese companies, especially Toyota, became dominant over competing companies from other countries. The reason behind Toyota's immense success lays behind the application of a lean method – the Toyota Production System (TPS). TPS is based on a people's natural attitude towards work, which manifests itself through their qualities, such as: the need for creativity, inability to comprehend distant deadlines, adapting to mechanisms of evaluation of their work, the need for personal contact and communication, the need to see and present results of their work, and aversion towards outer control [1].

As a form of organized work process, software development did not remain immune to the phenomena of agility. In a historical perspective, many techniques that have been used ever since the evolvement of the earliest development processes encompassed certain agile elements. The end of the past century brought some of the today's most frequently used development processes, which are completely based on agile values and principles. These processes, i.e. their scale-up, aimed at confronting challenges they face today, represent the topic of the research presented in this paper.

For the purpose of further clarifying the research topic, the terms most frequently used in this research are defined in the following text. In the scope of this paper, the term 'traditional development' denotes plan-driven development processes, or so-called heavy weight development processes, whereas the term 'agile processes' denotes a range of agile development processes, such as Scrum and XP, which fully incorporate the principles and values proclaimed in the 2001 Agile Manifesto.

Even though it has been a decade and a half since the Agile Manifesto was published, the popularity of agile software development processes has not waned. However, they are nowadays facing major challenges. Increasingly frequent changes in business requirements and the growing complexity of circumstances behind these changes are further complicated by the divergence of physical and logical aspects of business. This necessitates more responsive adaptation of information systems, which have consequently become more heterogeneous and decentralized [2]. Complexity of a system is determined by three main attributes: *scale*, *diversity*, and *connectivity*. 'Scale' reflects how many things are there in the system, 'diversity' is determined by the variety of things in the system, while 'connectivity' corresponds to how many different relationships are there between things [3-4].

Kruchten presented a contextual model for software-intensive systems development, intended for guiding the adoption and adaptation of agile software development practices. The model proved to be effective in cases when the project context was substantially distanced from the "agile sweet spot", i.e., the context from which the agile development stemmed from, and in which it is most successful [5]. Kruchten describes the "agile sweet spot" as "collocated team, of less than 15 people, doing greenfield development for non-safety-critical system, in a rather volatile environment; the system architecture is defined and stable, and the governance rules straightforward" [6].

Similar to Kruchten, Ambler defined eight scaling factors, which influence the complexity of a system and the environment it is developed in [6]:

- Team size – from under 10 developers to hundreds of developers
- Geographical distribution – from co-located to globally distributed
- Compliance – from low risk to critical/audited
- Organization and culture – from open to entrenched
- Organization distribution – from in-house to third party
- Governance – from informal to formal
- Application complexity – from simple, single platform to complex, multi-platform
- Enterprise discipline – from project focus to enterprise focus

As the project domain shifts from the "agile sweet spot", low ceremonialism and high iterativity, as the key characteristics of agility, no longer seem to be perceived as panacea. In addition to that, there is an apparent trend of combining complementary

elements (considered conflicting, until recently) of agile and traditional development, which proved their coexistence and integration possible [7-10]. Having abstracted their details, Matkovic et al. [7] provided comparison of the Rational Unified Process, XP, and Scrum development models. After the analysis aimed at finding optimal balance between iterativity and ceremoniality, the authors proposed a combined model that would encompass the advantageous, and exclude the obstructive features of RUP, XP and Scrum.

Architectural considerations are among the most sensitive issues when considering scaling up agile processes. Agile processes do not offer typical, explicit software architecture development activities, such as analysis, synthesis and evaluation, since they are believed to incur additional costs, and not create value for the user [11]. Generally speaking, there are two extreme architectural strategies: Big Design Up Front (BDUF) and emergent design. Supporters of agile development believe that the concept of metaphor, together with refactoring techniques, represent adequate substitutes for the traditional architecture development process. According to them, architecture is developed gradually with each iteration, as a result of continuous changes to the source code (emergent architecture) [11-13]. However, not denying that agile processes offer organizations efficiency, quality and flexibility in change management, several authors [8, 14-16] consider that explicit architectural practices have an important role in the development of complex software solutions. According to them, refactoring, as the architectural practice of agile development processes, can be successful enough only if the high-level design of software architecture was completed properly. This is the only way to avoid high amount of refactoring, which would cause an escalation of development costs in later stages, as well as erosion of the architecture, possibly jeopardizing the whole project [15, 17].

In this paper, the authors have explored scaling up of agile processes through use of elements typical of traditional development, referred to as "traditionalisation of agile processes".

Starting from the assumption that establishing a balance between agile and traditional development of software architecture, or more precisely, between explicit architectural practices and the agility of the development process [18-20] would ultimately facilitate overcoming the challenges agile processes face in the development of highly complex systems, this paper investigates the following research questions:

- **RQ1**. What was concluded in prior studies on the need for integrating explicit architectural practices into agile development processes?
- **RQ2**. Which architectural problems appear in the development of complex information systems using agile development processes?
- **RQ3.** How significant are particular, explicit architectural practices to agile development processes and the development of complex IS?

The remaining of this paper is organized as follows. Section 2 describes the research methodology, both of the theoretical and the empirical component. Section 3 describes the theoretical background with an overview of results of the previous studies included in the systematic literature review. Section 4 presents the results of the empirical research: qualitative, obtained through analysis of semi-structured interviews with domain experts, and quantitative, obtained by surveys. Section 5 outlines the threats to

validity and directions for further research. Finally, Section 6 contains a discussion on the research results, along with a comparison with related studies.

2. Research Methodology

The research design was developed by adapting the framework proposed by Hevner, March, Park, and Ram [21]. The sequence of research activities and the techniques used are presented in Figure 1. The research problem and the research questions stated in the first chapter are the results of the "Research subject identification" phase. Activities "State of the art exploration" and "State of the practice exploration" within the "Background research" phase provided answers to RQ1 and RQ2, respectively. "State of the art exploration" was conducted by means of a systematic literature review, in accordance with the recommendations provided by Kitchenham [22] (more details in Section 2.1), while guidelines provided by Miles and Huberman [23] served as a basis for interview coding and thematic analysis within the "State of the practice exploration".

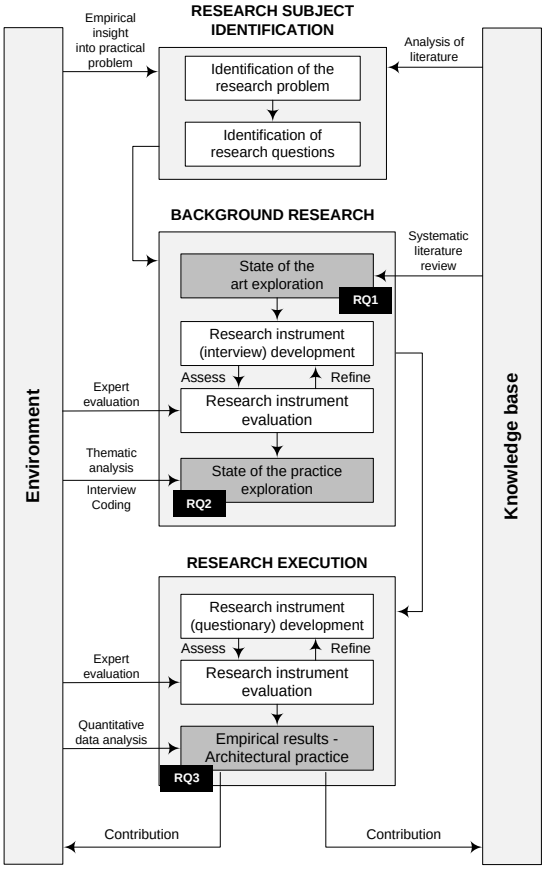

Fig. 1. Research methodology (Source: adapted from [21])

The final phase of the research encompassed a set or research activities, grouped within a logical section titled “Research execution”. Just as the interview used in the “Background research” phase, the questionnaire developed within the “Research instrument development” activity was refined with acknowledgement of the feedback provided by experts who subjected it to content validity analysis in several iterations (“Research instrument evaluation” activity). Finally, within the activity “Empirical results - Architectural practice”, qualitative analysis of data gathered through empirical research resulted in a set of explicit architectural practices, which provided an answer to RQ3.

2.1. Systematic Literature Review

The systematic literature review was based on the framework developed by Kitchenham [22]. The framework comprises three phases: planning the review, conducting the review, and reporting the review. The stages within planning the review are: identification of the need for a review, and development of a review protocol. Stages within conducting the review are: identification of research, selection of primary studies, study quality assessment, data extraction & monitoring, and data synthesis. ‘Reporting the review’ is a single stage phase, and an overview of its output is given in the Theoretical background section of this paper.

The defined research protocol required a strategy on which the search for primary research material would be based on. Specifically, the strategy involved:

- Definition of keywords for the search – Agile software architecture, Agile methods and architecture, Agility and architecture.
- Selection of sources for the search – the Web of Science and SCOPUS bibliographic databases were chosen as the most prominent sources of scientific and professional papers.
- Definition of criteria for inclusion/exclusion of research material – research and professional papers published in reviewed journals and conference/workshop proceedings between 2000 and 2014 were deemed acceptable, while all papers that did not associate the term ‘agility’ with agile development processes, papers that were not based on empirical research or did not have a valid approach/method, as well as papers based solely on experts’ opinions were excluded from the analysis.
- Evaluation of quality of the research material compliant with the previously defined criteria for inclusion was carried out in accordance with criteria proposed by Dyba and Dingsoyr [24], while the extraction and synthesis of key information from the relevant research material was performed by use of the NVivo software suite, for easier management of concepts, findings and conclusions contained within the analyzed papers.

In accordance with the recommendations by B. Kitchenham, the systematic literature review process started with the initial search for primary research material; Web of Science and SCOPUS were used to identify papers with relevant associations to the defined keywords. The result of the preliminary search through Web of Science and SCOPUS was a set of research and professional papers published in reviewed journals and conference/workshop proceedings, selected in accordance with the defined

keywords. The identified papers were accessed through the following electronic services: IEEE Xplore, ACM Digital Library, and ScienceDirect. Springer was, among others, excluded in this phase, since the papers that met the criteria for inclusion had already been found in the sources listed in Table 1.

After the preliminary analysis described above, the sources listed in Table 1 were queried in accordance with the protocol defined for the systematic literature review. An overview of the total number of hits per each electronic service is given in Table 1. The research resulted in 34 relevant papers (out of 69 potential ones), 26 of which were the result of the primary search, while 8 were the result of the secondary search. Secondary search denotes an analysis of references provided in the primary research material.

With the defined research protocol, the authors narrowed down the focus of the analysis to academic papers directly related to the research questions. Books were not included in the systematic literature review; however, several were referenced in other sections of the paper: Introduction, Related Work and Discussion, as well as in the Concluding Remarks.

Table 1. Search results per each electronic service

Source	Number of hits with the defined keywords	Number of papers selected for further analysis	Number of papers excluded
IEEE Xplore	701	45	656
ACM Digital Library	237	12	225
ScienceDirect	46	12	34
Total	984	69	915

2.2. Development and Evaluation of Instruments for the Empirical Research

Empirical research included both a qualitative and a quantitative component. Accordingly, two research instruments were developed: a questionnaire for conducting the interview and a questionnaire for the realization of the survey.

An initial set of questions for a semi-structured interview with expert practitioners was defined for the purpose of answering the second research question, through State of the practice exploration. The semi-structured interview method was selected with the intent of collecting as much information on the research context and practical problems as possible. The initial interview questions were generated on the basis of analyzed of research materials.

The second research instrument–questionnaire for the realization of the survey–was created with regard to the previously completed systematic literature review and qualitative analysis of interview data. The questionnaire was developed in an electronic form, using Google Forms. Chosen expert practitioners were asked to answer a set of closed questions, given in the form of assessment scales (modeled consistently with the Likert-type scale) and checklists. Quantitative analysis was carried out on collected data for the purpose of answering the third research question (Empirical Results–Architectural practice).

Evaluation of research instruments (interview and survey) was carried out by a group of experts in the area of agile development and software architecture: three expert practitioners and two researchers with a PhD in this area. Each potential question was evaluated using a Likert-type scale: 1 - not significant; 2 - somewhat significant; 3 - significant; 4 - extremely significant.

Following the evaluation, content validity index was calculated for each question, as well as for the whole interview and the survey, in accordance with the procedure prescribed by Polit and Beck [25]. The value of the content validity index for the first version of the interview was 0.76, which indicated that it was necessary for it to be amended, in agreement with experts' input. Amendment of the interview involved elimination of some questions with validity index lower than 0.8, reformulation of certain questions, merging of particular questions into a single question, etc. The content validity index for the entire survey was 0.83. The final version of the questionnaire did not include questions with values of the content validity index lower than 0.8.

The final version of the interview contained 40 open-ended questions, divided into 5 thematic areas: (1) Data on the respondent and context, (2) Data on development process models, (3) Data on identified problems and their causes, (4) Data on software architecture, and (5) Contextual factors. The final version of the questionnaire included 30 closed-ended questions.

2.3. Empirical Research

This section contains descriptions of respondent sampling, means for gathering empirical data, and methods used in quantitative and qualitative analysis of data.

Selection of Respondents. The nature of the research problem necessitated purposive selection of sample units ($n \geq 20$) to permit the application of both instruments developed for the empirical research. Purposive sampling was necessary to avoid including individuals who lack required type and quality of knowledge, skill, expertise, experience and information from the problem area. The research was conducted in prominent companies within the IT sector, on a "purposeful", homogenous sample of 20 Serbian experts. The same panel of respondents was used both for the interview and the survey. Respondent recruitment was based on a defined list of criteria that the potential participants were required to meet. The list was created by modifying the general criteria defined by Skulmoski, Hartman and Krahn [26] and Ziglio [27]:

- Knowledge and practical experience in the development of software architecture and complex systems using agile processes
- Capacity and willingness to contribute in the research
- Confirmation that they will devote sufficient time and be dedicated to the research
- Good communication skills
- Academic education in information technology
- More than 5 years of professional experience

Additional, bootstrap sampling with 1,000 replications was carried out in the IBM SPSS Statistics suite, in order to ensure stability of the research results.

Collection and Processing of Empirical Data. Interviews were conducted "face-to-face" and recorded (with respondents' consent), as to ensure better accuracy and completeness of data. Data gathered during interviews was transcribed with a word processor and subsequently subjected to qualitative analysis in the NVivo software suite, using the key-word-in-context (KWIC) technique [28] and thematic analysis of the content [23, 29-31]. The qualitative analysis resulted in the identification of categories of practical architectural problems, which represent an answer to the second research question.

As mentioned previously, the survey was carried out electronically, via Google Forms. The participants received a link to the questinnaire by email, along with the instructions on how to complete it. Participants were guaranteed anonimity, data confidentionality, privacy, and fair use. Data acquired via the questionnaire was imported from Google Sheets into MS Excel, where it was prepared for quantitaive analysis in the SPSS software suite. Several techniques were used in the quantitative analysis, namely: descriptive statistics procedures, Efron's resampling, and Cohen's kappa coefficient for assessing respondents' agreement. Bootsrapping was used in order to ensure stability of results of the empirical research. To be exact, results from the empirical sample of 20 experts were more statistically significant to conclusions related to the overall population due to software bootstap sampling with 1,000 replications. The qualitative analysis resulted in a set of explicit architectural activities that expert respondents rated as significant for the development of complex systems and suitable for incorporation into agile software development processes. This provided the answer to the third research question.

3. Theoretical Background

Claim made by supporters of agile development that explicit architectural practices are unnecessary in agile processes is the research subject of the majority of studies included in the systematic literature review. The literature analyzed suggests that emergent architecture may be a viable alternative to conventional approaches to software architecture development, but only in some architectural areas, while being completely inadequate for others. Friedrichsen [32] states that emergent architecture is a good practice for detailed design, but that it does not cover a set of important architectural activities that are supposed to answer whether a solution is doing what it is supposed to do. These activities involve communication with stakeholders, aimed at gaining insight into their needs, identifying requirements, and overcoming contradictions and conflicts between the identified requirements. Elimination of this explicit architectural activity increases the actual risk of wrong decisions made in the design stage remaining unseen until it is too late [32].

Findings of the literature overview also suggest that non-functional requirements are not given enough attention in the design process. This is often justified by the fact that implementation of non-functional requirements is carried out afterwards, through changes in the source code during the maintenance, since maintenance lasts longer and has a larger budget [18]. However, Bellomo, Nord, and Ozkaya [33] believe that in the case of large-scale and complex systems development, conventional agile process should be extended as to include the following explicit architectural activities related to

non-functional requirements: test driven development with focus on non-functional requirements, prototyping with focus on non-functional requirements, and technical debt monitoring with focus on non-functional requirements. Cleland-Huang, Czauderna, and Mirakhorli [34] introduced the notion of architecturally savvy persona, who is in charge of identifying and analyzing non-functional requirements. The same authors intended to enhance the process of discovery, analysis and management of architecturally significant requirements by introducing the concept of personas from various domains.

Jeon, Han, Lee, and Lee [35] have proposed the Acrum method, which extends Scrum with three explicit architectural activities focused on non-functional requirements: 1) analysis and management of non-functional system requirements (subsequent to analysis of functional requirements); 2) mapping functional and non-functional requirements using relation association matrix that represents their correlation, in order to ensure traceability and completion of both product and sprint backlog; 3) verification of fulfillment of non-functional requirements after each sprint. If the verification process shows that a previously identified non-functional requirement has not been fulfilled, even if all functionalities it is associated with had been implemented, these functionalities must be revised, or a new strategy for fulfilling the given non-functional requirement must be formulated.

Brown, Nord, and Ozkaya [36] also highlighted the necessity for explicit identification of non-functional system requirements, as to support discovery of dependencies between functional and non-functional requirements and architectural elements in each iteration. Both functional and non-functional requirements need to be prioritized, so that a proper schedule may be defined for each release. Faber [37] also states that it is the architect's responsibility to deliver non-functional system requirements as value to users, as well as to implement them in close cooperation with programmers.

Explicit architectural activity of defining architectural structures is not included in emergent architecture [32], as is the case with anticipation of future system changes, which is an activity critical to the decisions on the time of realization of particular architectural activities, based on cost/benefit analysis [36]. Architectural planning involves architectural considerations that go beyond a current iteration, aimed at anticipating future requirements that the architectural solution should support [32, 36, 38].

Planning that is limited to one iteration leads to design degeneration and loss of flexibility, which may hinder the agility of the whole project [39]. The main reason for this is that functional requirements cannot be analyzed and developed completely separately, since they are interdependent [36]. Functional requirements with high business value to the user, and accordingly, high priority, often depend on requirements with lower business value that need to be implemented first. In addition to analyzing interdependencies between functional requirements, it is also necessary to analyze dependencies between functionalities on one side, and non-functional requirements and architectural elements of the system on the other. Otherwise, the risk of implemented design decisions being inadequate increases, which may lead to an increase of technical debt in the future. Proliferation of technical debt over time causes problems that cannot be solved only through modifications of the source code, but rather require radical changes in the architecture [36, 40]. In addition to selection of functionalities that should be implemented within an iteration, the suggested concept extends release

planning as to include identification of architectural elements that need to be developed to support functionalities and future changes [33, 41]. Placing all design activities within the present iteration is an extremely hazardous strategy, especially in software development projects within large business organizations, characterized by a great number of different applications (legacy and novel), various technologies and a great number of teams [38].

Weitzel, Rost, and Scheffe [39] have coined the term "epic architectures" to denote widening the scope of architectural planning beyond one iteration in Scrum. Epic architecture is an architecture designed for a coherent group of functional requirements. The aim of epic architecture is to define common elements. It is developed for around 8 planned sprints. Recognition and implementation of their similarities in the current sprint reduces the total effort necessary for implementation, while simultaneously increasing the uniformity of functionalities that need to be implemented in subsequent sprints. Implementation tasks for a sprint are derived based on defined architectural requirements, which make up architectural stories.

According to past studies, establishing a balance between extensive architectural planning and emergent architecture still represents a challenging issue. Analysis of papers dealing with this issue reveals a common stance that up front design must be adequate in terms of such balance, which means that the trap of BDUF strategy, typical for traditional development, must be avoided. The following text contains an overview of perspectives on this issue, encountered in the analyzed literature.

Friedrichsen [32] points out that an estimate of adequate extent of up front analysis and design depends on software architects' experience, skills, knowledge, as well as effective communication with stakeholders, while Waterman, Noble, and Allan [42] added two more factors – understanding of the selected architectural solution and use of a predefined architecture (in terms of existing patterns, architecture recommended by vendors or tools used to automatize the process). Brown et al. [36] also advocate that architecture planning must not be too extensive, but rather "sufficient". They proposed an approach based on three concepts: dependency analysis, real option analysis, and technical debt management.

Dependency analysis involves examining and managing dependencies among functional requirements, dependencies between functional and non-functional requirements, as well as dependencies between requirements and architectural elements. The aim of dependency analysis is to facilitate timely development of architectural elements that can support implementation of necessary functionalities. This requires architecture planning that extends beyond one iteration, i.e., anticipation and analysis of future needs. Analysis of both kinds of requirements is represented using a single table. After that, DSM (Dependency Structure Matrix) analysis is executed to reveal dependencies between all constituent subsystems that represent certain functionalities (e.g. exchange of data between sales and purchasing subsystem), followed by DMM analysis (Domain Mapping Matrices), to determine their dependence on particular architectural elements (such as user interface components, data access procedures, security, etc.). The importance of software architecture analysis process in agile processes was also emphasized by Buchgeher and Weinreich [43], who concluded that the most effective technique is once again dependency analysis, but on the code level. The focus of the proposed technique is on detecting static dependencies from the source code, as to compare implemented architecture with the planned. Identified dependencies are used as an indicator of relations between these two levels of architecture, which,

along with standard agile practices (continuous testing, continuous code analysis, continuous code integration, continuous refactoring, and pair programming), facilitates additional continuous quality control.

Once the dependency analysis is finalized, Brown et al. [36] consider optimal choice of necessary architectural elements to be of key importance to release scalability. For this purpose, the authors suggested using the real option theory – financial analysis model used in corporate finance to assess cost-effectiveness of particular business decisions. Real option analysis can be used effectively in release planning, for allocating architectural elements to specific releases. Real option analysis and technical debt management can optimize investment in particular architectural decisions, based on results of dependency analysis and cost/benefit analysis of the architectural decision in question. The ultimate decision should also be justified from the point of mitigating risk associated with future uncertainties. The goal is to reach a suitable and cost-efficient solution today, without jeopardizing the possibility of developing a more complete solution tomorrow.

Real option theory is also used by Blair, Watt and Cull [44], but for solving the problem of finding the right time to make architectural decisions, since they believe that architectural decisions in agile processes are made either too early or too late. They proposed a framework that should guide teams in recognizing the most suitable moment for making particular architectural decisions. The framework involves using a spreadsheet in which development phases are listed in columns, while architectural issues constitute rows. The proposed framework is aimed at keeping the front up design within certain limits, as to avoid the BDUF trap. Ven and Bosch [45] were also concerned with improving the process of architectural decision-making, starting with the assumption that architectural decisions in agile projects are made just-in-time by programmers, while the architect has a consultative role in this process. The authors presented the 3A framework (agile, architecture, axes) based on three axes that need to be considered in order to establish a uniform process for architectural decision-making that would be appropriate for a particular project. The first axis (who) contains roles with potential responsibilities in the process of architectural decision-making (development team, application architect, domain architect, enterprise architect, management). The second axis (how) contains means of documenting architectural decisions for communication with the stakeholders (direct communication, meeting notes, ad-hoc documentation – wiki, informal documentation based on templates and formal documentation based on templates). The third axis (when) contains different periods from the moment an architectural decision is made to the moment the feedback on its validity is received (more than 6 months, 1-6 months, less than 1 month).

Kruchten [46] suggested a set of heuristics for solving the problem of balancing functionality development and software architecture. Implementation of proposed heuristics requires a workshop with various roles in the project, aimed at examining the eight suggested dimensions: semantics, scope, lifecycle, roles, documentation, method, value, and cost. By answering key questions in these areas, participants develop a common mental model concerning the application of certain architectural practices. After that, they are able to define the management process, as well as the technical process, which steers architectural activities within an agile process (so-called “zipper-model”). Chen and Babar [47] constituted four categories of contextual factors (project, team, practice and organization) that determine whether efficient architecture can be developed solely through modifications to the source code. The proposed framework is

complementary to Kruchten's contextual model developed based on experience. Chen and Babar [47] extended Kruchten's model with empirically identified factors, such as experience and skill.

Hadar and Silberman [48] proposed a C3A model, which interprets the concept of agile architecture as a synthesis of so-called reference architecture, which illustrates the vision in technical and functional terms (it is a result of planning), and implementation architecture, which scopes parts of the reference architecture into a future release (development of functionality). The method includes a process for evaluating reference architecture and evolution of implementation architecture, used to analyze the gap between what was planned and what was accomplished within a release. Key steps of this method include: listen and observe, watch, reflect, improve, scrutinize, and kick-start.

Several authors investigated the role of software architects in accomplishing a balance between up front and emerging architecture in agile development processes.

Faber [37] considers that changing the conventional role of architects is imperative for balanced investment into functional/non-functional requirements. He underlines that architects are responsible for overall quality of the system and that their choices of adequate design decisions affect the balance between implementation of functional and non-functional system requirements. Architects are supposed to be service providers to both programmers and clients, having multiple roles in interaction with them. They should provide value to clients through implementation of non-functional system requirements, as well as provide continuous support to programmers throughout the course of implementation.

Hadar and Sherman [49] also stated that is important to include the software architect into the entire agile development process, while Blair et al. [44] stress the need for the architect's close collaboration with the development team, with continuous exchange of ideas throughout the whole project. Hopkins and Harcombe [50] believe that the software architect's role is indispensable in the development and delivery of large, complex systems, which typically necessitate development and integration of multiple systems and coordination of hundreds of individuals. According to them, the software architect is the only person who makes decisions on vital aspects of the system, since even senior programmers are often unable to do so. The role of a software architect is to examine the problem being solved from different perspectives at the very beginning of the project, since each business problem is unique and requires different approaches to identifying its distinctive aspects. After the conceptual description of the system is completed by the software architect, the agile team can decide on how to test it (e.g. dynamic execution, static testing, or simulation). With early instrumentation and validation of this process, the architect is supposed to turn their vision into a common one (of the whole project team), and to monitor development, as well as to react in case of unexpected problems or necessary changes.

Madison [51] believes that software architects should be ones to close the gap between the agile development process and methods for developing software architecture. In order to develop agile architecture that balances both the traditional and the agile approach, software architects must have exceptional understanding of the agile process, as well as the ability to create a balance between business and architectural priorities. Madison presented a hybrid model of Scrum, XP, and sequential project management for developing agile architecture. He advocates use of architectural functions and skills (communication, non-functional requirements, choice of suitable

software and hardware, design patterns) throughout the four stages of the process: up front planning, storyboarding, sprint, and working software.

Hopkins and Harcombe [50] point out that it is necessary to establish a balance of up front architectural activities, as to avoid jeopardizing the concept of agility on large-scale and complex projects. They propose that every project should start with up-front risk analysis, in order to identify and isolate complex areas, and identification of software elements, infrastructure, and data architecture. Start of the project implies a description of the problem on a conceptual level, along with suggestions of solutions to problems within the given area. Occasionally, the problem/solution is reached through analysis of existing systems, or by opting for a commercial solution for certain areas. As early as possible, the architect should also decide on how the riskiest aspects of each identified problem will be tested, identify architecturally significant requirements and, if necessary, develop prototypes.

For medium and large development projects, Qureshi [52] suggested modifying and extending development stages of the XP process: project planning, analysis and risk management, design and development, and testing. Nord and Tomayko [8] proposed a hybrid model for large and complex agile projects based on integrating the XP process with methods for developing software architecture developed by the Software Engineering Institute of the Carnegie Mellon University (architecture tradeoff analysis method, quality attribute workshop, attribute-driven design method, cost benefit analysis method, active reviews for intermediate design). These methods can add value to agile processes, since they accentuate non-functional requirements and their significance in the architecture design.

Nowadays, decentralization, heterogeneity and the need for interoperability are among the most important challenges that business are faced with. Development of such systems usually involves many team members, while the systems often must be scaled to highest levels of performance and security. They are often mission-critical, and it is understood that they must not fail. All this necessitates a strong architectural support to the system and appropriate documentation. On the other hand, users expect these software solutions to be adaptable to changes in the business environment, which requires application of principles of agile development. Boehm, Lane, Koolmanojwong and Turner [41] identified three factors that need to be observed in establishing balance between architecture and application of principles of agile development in complex system development projects: system's size, criticality, and requirements volatility. They suggested an approach based on quantitative appraisal of costs and risks (by use of COCOMO II model and the concept of risk resolution factor (RESL)) of investments into architecture. Based on research results, the authors suggested a hybrid (agile/plan driven) process framework for developing such complex systems - incremental commitment model (ICM). The framework represents a synthesis of concepts from the existing process models: the concept of early verification and validation from the V-model, concurrency concepts in the concurrent engineering model, lighter-weight process concepts from the Lean, and other agile models, risk-driven concepts from the spiral model, as well as the phases and anchor points from the Rational Unified Process (RUP).

Nord, Ozkaya, and Sangwan [40] proposed a way for establishing a balance between up-front and emergent architecture through application of the Lean concept of managing the value flow throughout the development process. According to the Lean concept, all the waste from architectural activities can be divided in three closely related categories:

overproduction, delay, and defect. Results of their research [40] have shown that incremental architecture development causes increases in costs associated with the aforementioned wastes. The research resulted in guidelines for improving the development process through more efficient time management and containment of waste in production (WIP). The authors believe that the strategy of developing the architecture in many small increments can reduce delay costs associated with the time required to complete the entire architecture design up front. However, refactoring the architecture can be much more expensive, since it may necessitate significant changes. Development in a smaller number of more extensive increments reduces costs associated with refactoring, but increases delay costs due to a greater amount of architectural activities. Delay costs arise either from waiting or delaying the implementation process, while refactoring costs arise from architectural flaws or waste caused by overproduction. These represent the two extreme strategies for iteration planning. In order for the development to take the intermediate path in terms of costs, the authors suggest utilizing the concept of visualizing investments into architecture within each increment, as to demonstrate the effects of architectural waste (due to overproduction, delay or defect) on the entire project. For this purpose, they suggest identifying architecturally significant requirements, as well as acceptance testing, in order to ensure visibility of architectural tasks in the backlog or the Kanban board. Utilization of the WIP concept in acceptance testing improves the flow of the development process, since it enables managing waste associated with over-architecting [40].

Hinsman, Sangal, and Stafford [53] point out that architecture visibility is crucial for establishing balance between up front and emerging architecture, especially in case of a volatile environment. They believe that the traditional code refactoring technique is not an adequate solution for problems arising from code complexity and side effects caused by rapid development, adaptation to business changes, and system upgrades. Instead, they propose a higher-level approach – architecture-based refactoring; a process guided by an architectural blueprint, which is a result of identified dependencies on a structural level. Refactoring is prototyped, prior to being applied to the source code. It involves five steps: define the problem, visualize the current architecture, model the desired architecture in terms of current elements, consolidate and repackage the code base, and automate governance of the architecture through continuous integration. Empirical data suggests that architecture-based analysis can enhance productivity of software development and reduce costs of system maintenance [53].

Stal [17] suggests that architecture refactoring should be included into agile development as a compulsory process, since that could enhance early detection and elimination of inadequate or suboptimal design decisions and ensure a high quality of architecture. This activity needs to be performed at least once per iteration. He suggests an approach for systematic execution of architecture refactoring process, which involves the following key steps: architecture assessment (identification of design problems); prioritization (determining the sequence for solving architectural issues, based on their significance); pattern selection (if they exist) for each identified issue, or conventional redesign; quality assurance, through assessment or testing, to ensure that refactoring does not cause accidental changes to the semantics. Architectural issues that should be solved using this process include: unclear roles of entities, complexity of the architectural solutions, excessive centralization, asymmetric structure or behavior, etc. If issues remain unresolved in a timely fashion, design erosion will occur and

refactoring will no longer be able to provide solutions to remaining architectural issues. In such case, the remaining solutions for mending the architecture are reengineering or redevelopment.

Keuler, Wagner, and Winkler [54] suggest ensuring visibility of architectural structures though application of a framework that links architectural decisions with the source code. The process begins with the description of the architecture in an XML file that is subsequently used to generate code that implements the described architecture, serving as the application's skeleton. The framework enables multiple teams to work on the same code with minimal conflicts, as long as dependencies within the source code are managed effectively. This enables programmers to deal with components–functionality, while the implemented framework manages components, interfaces, and their dependencies.

Results of the literature review suggest that a great portion of problems arises from one additional essential conflict: requirement of minimalism in agile processes and the need for well-documented architecture in complex systems [49]. Evidences from the industry reveal that, in most cases, architectural documentation is either excessive, or completely absent [55]. Inadequate documentation results in evaporation of architectural information and knowledge [48], poor understanding of the architecture and impaired communication, which leads to chaos and project failure [56]. Excessive documentation causes waste, in terms of time and resources, as well as straying from the essence [56]. Faber [37] 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. While Fallesi et al. [57] published empirical findings suggesting that agile programmers find architectural artifacts useful for easing communication between members of the development team in latter stages of design, as well as for documenting and assessing the solution, Babar [18] reached empirical findings that suggest that a modified traditional documenting practice, Software Architectural Overall Plan (SAOP), is commonly used, but only for a conceptual description of the architecture, whereas all other design decisions are described in the wiki.

The literature contains several suggestions on how to overcome the described problem of documenting. Tyree and Akerman [58] see the solution in documenting of architectural decisions and their clarifications. Hadar, Sherman, Hadar, and Harrison [55] suggested a template for documenting software architecture that is in line with the agile philosophy and Lean documentation. Eloranta and Koskimies [59] suggest applying the Architecture Knowledge Management (AKM) concept, which they modified and integrated with the Scrum process. This approach involves development of an architectural database and application of a decision-based architecture evaluation method.

Results of the theoretical research suggest that there are quite a number of architectural issues that various authors are concerned with. An overview of key categories of architectural issues and authors that investigated them is given in Table 2.

Table 2. Architectural issues identified in the literature (Source: Authors)

Current architectural issue	Source
Non-functional requirements	[18, 33-37]
Anticipation of future requirements and envisioning architecture beyond the current release	[32-33, 36, 38-41]
Balance between up front and emergent architecture	[8, 32, 36-38, 40-48, 50, 52-53]
Software architect's role	[18, 37, 44, 49-51]
Visibility of architectural tasks	[17, 53-54]
Architecture documenting	[18, 37, 48-49, 55-59]

4. Results of the Empirical Research

Results of the empirical research are presented as (a) an overview and description of qualitative results obtained through the analysis of semi-structured interviews with domain experts, and (b) an overview of results of the quantitative part of the research, as a set of explicit architectural activities significant to agile development.

4.1. Interview Results

Over 70 topics related to practical architectural issue were identified based on transcripts of interviews with 20 experts in agile development, coded in NVivo software suite. Similar topics were combined into concepts, and similar concepts were divided into 8 categories of practical architectural issues (Table 3). Identified categories of practical architectural issues indicate the necessity of applying certain explicit architectural practices in the development of software architecture in agile processes. The identified categories represent the answer to the second research question.

Table 3. Identified categories of practical architectural issues (Source: Authors)

Category	Concept
Functional requirements	1. Incomplete requirements 2. Volatile requirements
Non-functional requirements	1. Inadequate consideration and identification of non-functional requirements 2. Inadequate monitoring of implementation of non-functional requirements 3. Not testing non-functional requirements 4. Neglecting refactoring, which should improve design quality 5. Delayed resolving of issues related to non-functional requirements 6. Technical debt

Vision of the Architecture	1. Lack of strategic architectural planning 2. Overlooking future system requirements 3. Inadequate allocation of time to research and analysis of architectural requirements 4. Overlooking certain aspects significant to architecture development 5. Neglecting the choice of an adequate architectural solution for prompter implementation of functionality
Technical and technological aspects	1. Failure to explore possibilities and limitations of current technologies, frameworks, and third-party libraries 2. Unbalanced application of traditional and novel technologies 3. Insufficient familiarity with the technology used 4. Inadequate choice of technology and implementation framework for the problem being solved
Business analysis and understanding of the problem	1. Insufficient time for business analysis process 2. Poor understanding of the problem 3. Lack of domain-specific knowledge
Architectural evaluation	1. Neglecting testing fulfillment of non-functional requirements 2. Infrequent prototyping, which should prevent bad design 3. Informal architecture review process 4. Neglecting metrics and tests 5. Rare use of time-limited proof of concept
Role of the software architect	1. Architects' coordinating role in detailed design 2. Some architectural decisions cause bottlenecks
Team	1. Inexperienced team members 2. Limited supply in the labor market
Architecture documenting	1. Inadequate management of architectural knowledge 2. Architectural decisions and the reasons behind them mostly remain undocumented

Comparison of architectural issues in the existing literature (Table 2), and architectural issues identified after the analysis of empirical data gathered through interviews (Table 3) shows many matches. In other words, issues addressed by researchers correspond to problems identified in the practice. The following text contains an interpretation of results given in Table 3, with references to respondents' statements. A code system is used to refer to individual experts (RSP1, RSP2…, RSP20) instead of personal names, as to ensure anonymity of respondents and their organizations.

Empirical findings show that requirements represent an important factor when it comes to a choice of an architectural strategy, since their traits influence strategic orientation in relation to the two extremes – BDUF and emergent – and therefore represent a source of numerous architectural issues. Most respondents stated that they operate in volatile environments, since clients often lack clear vision of what they need, which is an additional cause for delays in the implementation phase (RSP10). This implies that requirements that the software architect operates with at the beginning of the project are incomplete (RSP8). Quality of requirements has a significant impact on

the amount of time the software architect needs to devote to up front architectural analysis, in order to identify the scope of the main part of the software. Lack of devotion to identification of architecturally significant requirements causes an increase in total project costs, as well as its duration (RSP8). *Requirement volatility* is the second most frequent problem that agile software architects and agile teams face. It is not rare for clients to completely change their idea on what they expect from the software solution during the implementation phase (RSP13). RSP6 believes that the best way to mitigate this risk is to put more effort into the initial phase of the project, before setting the architectural solution for the main part of the system. In line with that, agile teams should identify the set or architecturally significant requirements for the main part of the system at the beginning of the project, leaving identification of other requirements and iterative enhancement of the developed architectural skeleton for the implementation phase (RSP9).

Practitioners also stated *identification of non-functional requirements* as a problematic area, due to the fact that stakeholders are mostly unaware of them, and put too much emphasis on the implementation of functional requirements, at the requirements. There is no systematic *monitoring of implementation of non-functional requirements*, as opposed to functional requirements, which are monitored via the Backlog and the Product Backlog. *Testing of non-functional requirements* is also not a consistent and mandatory practice, as is the case with functional requirements. This is the reason why timely refactoring, *which should improve design quality,* is neglected in practice; instead, the design is "patched up" each the team needs to overcome a burning issue and enable the implementation of subsequent functional requirements. Such approach implies *delayed resolving of issues related to non-functional requirements* and constant presence of *technical debt*. The solution suggested for this problem relies on creating and continuously updating a single prioritized list of functional and non-functional requirements and displaying all tasks on the Kanban board (RSP20).

The second group of architectural issues is related to the vision of the architecture. When it comes to the selection of an architectural solution, the most common approach in the industry is "cognitive exploration" by the architect, as well as brainstorming with other architects and members of the development team (RSP9). When opting for a solution, architects mostly rely on their personal experience and knowledge, as well as expertise of the development team (RSP2, RSP3, RSP4). Due to time constraints, *the selected architectural solution is often not the most suitable one for the problem* being solved. In most cases, the client bears responsibility for the choice of an architectural solution (RSP3). The practitioners noticed that they need more time for *research and analysis of architectural requirements* and confirmation of architectural solutions. In addition to that, they are aware of the fact that, in their attempts to initiate implementation as soon as possible, they fail to consider all *significant architectural aspects* at the beginning of the project (e.g. deployment), as well as the *future system requirements*. This implies a series of consequences and greater costs in latter development phases. In actual fact, there is no *balance between the strategic and tactical architecture planning*, that is why there is a risk of taking wrong turns in the development. This increases the risk of architectural erosion. In addition to the fact that changing major architectural decisions in the latter phases of development is costly, the scenario in which the architecture cannot be mended by refactoring also remains a possibility.

Practical results revealed a category of architectural issues related to technical and technological aspects of architecture development. RSP15 provided the most vivid portrayal of these problems, further claiming that architectural decisions need to be made with regard to where and how the software product will be executed. RSP1 even stated technological aspects as a key factor in architecture development, and that it is therefore necessary to devote enough time to explore the *possibilities and limitations of current technologies, frameworks, and third-party libraries* at the beginning of the project. This is important for an *adequate choice of technology and implementation framework*, with regard to the problem being solved. RSP2 and RSP5 stated that it is equally important that team members, especially the software architect, are *familiar with the technology used*: latest trends in architectural options, technological innovations, and third-party components that can be used in the development. In addition, the respondents stated that there is a trend of using novel technology, without clear arguments in favor of such choice. Explanations are based solely on the argument that 'new' is better than the old. This way of thinking is wrong, and should be replaced with *balanced application of traditional and novel technologies*, with regard to their advantages in solving the problem.

The respondents recognized *insufficient time for business analysis* and problem examination as one of the serious problems, which results in inadequate architecturally significant requirements. Time spent on analysis is directly proportional to the level of *domain-specific knowledge* of the problem that the team members, above all, the software architect, are dealing with.

Practitioners think that it is critical for the success of a software architecture that the architect explores the problem from different perspectives at the very beginning of the project, since every business problem is different, with unique architectural aspects. The most important thing for solving a business problem is to reach the *understanding of the problem* by comprehending how the target organization operates (RSP13), that is, its business processes, since they represent a basis for identifying architecturally significant requirements (RSP12). Gaining understanding of business processes relies on an explicit architectural activity, which involves discussion with key stakeholders in group meetings or individual interviews. The respondents have pointed out the importance of software architect's involvement in the meetings with the product owners, since otherwise they never obtain all necessary information from the documentation (RSP7). Although product owners are often (but not necessarily) technical personnel, they do not have the same level of knowledge and experience as software architects, and therefore cannot convey 100% of their requirements (RSP16).

Architectural evaluation implies verification of architecture in relation to architecturally significant requirements. Results suggest that this segment of architectural issues is mostly based on an ad hoc approach. Besides the standard agile practices (code review, code integration, regression testing, static and dynamic code analysis, etc.), the respondents are aware of the significance of several traditional practices (*prototyping*, *time-limited proof of concept*, *formal review by experts* in architecture), but they also stated that usually lack time for these techniques and that they apply them only when it is necessary. Practitioners are also aware of the fact that *metrics and tests* are the best means for reviewing architecture, that is, fulfillment of non-functional requirement (RSP20), and that they aspire to using them. However, they also stated that they still review architecture ad hoc, without a defined process. *Testing fulfillment of non-functional requirements* is most often allocated to the maintenance

phase, as to expedite delivery of value to users through development of functionality, or because of budget limitations.

The formal software architect role is not standard in agile processes. Agile teams are cross-functional, and all team members share the responsibility for the architecture. However, agile teams included in the research usually have a formal software architect role, performed by a highly experienced programmer (RSP5). As an experienced programmer, the software architect participates in the development team at the beginning of the project, as to set up the main part of the software with the programmers (RSP5). As several authors agree, the role of a software architects in an agile team should significantly differ from the traditional one, because architects must continuously be engaged throughout the whole development process. Their role must involve coordination throughout the whole development of the software solution [44, 49-51]. However, the respondents stated that the *architect's only role is the coordination of detailed design.*

In order to achieve the coordinating role of a software architect throughout the whole development process, practitioners suggest the strategy aimed at *"...raising awareness, trust, skills and knowledge from the problem domain and technology..."* among all team members. Improvement of team members' level of technical knowledge can be achieved most efficiently by including them in discussions on architecture when it is first developed at the beginning of the project, as well during iteration planning. By following this approach, teams can avoid *architects being the bottleneck*, in case a radical change in design occurs due to an architectural decision. In such way, other team members who would otherwise lack even the basic architectural knowledge, or a whole picture of the solution, may help expedite the implementation of the architectural decision (RSP20). Agile teams are aware that the responsibility for the architecture should lie on the whole team, not just on the architects. However, such approach requires skilled individuals, with high level of technical knowledge, which is the main problem that agile teams face in practice. Rapid development of IT industry in Serbia on one side, and *limited supply in the labor market* on the other, lead to the fact that agile teams consist mostly of *inexperienced individuals*. Causes of this problem identified in the research involve the constant inflow of new employees, and the fact that an average engineer in a team is at the junior level (RSP20).

Documentation is a very important architectural issue. Traditional development overemphasizes this activity, while agile processes nearly replace it with the idea of source code as the ultimate documentation, in line with the proclaimed agile value of "working software over comprehensive documentation" [60]. However, when it comes to development of complex software solutions, quality source code is not a sufficient documenting practice; it is important to incorporate some of explicit documenting practices in moderation. Architectural documentation is mostly written in wikis, and contains descriptions of architecturally significant functional and non-functional requirements, as well as the decision on technology stack (RSP4, RSP9, RSP11). It also includes basic architectural models, hand-drawn in form of flowcharts. Formal models are rarely used in practice, since they require a lot of time and effort for continuous revising, which reduces agility. However, the problem is that there is *no documentation for most architectural decisions and reasons behind them.* As an outcome of such absence of strategy for *managing architectural knowledge*, most of the architectural knowledge remains "trapped" in individuals' minds, and is therefore impossible to reuse it. Agile teams recognize this problem, but still lack a solution for it.

4.2. Survey Results

Collected data was subject to quantitative analysis, with the aim to determine significance of explicit architectural practices to agile development processes. Quantitative analysis was conducted for each architectural practice (of 31 in total), and an example of an analysis for a particular agile practice – *identification of key system stakeholders* – is given in the following text.

Frequencies of significance scores of the observed architectural practice are presented in Table 4, both in numbers and percentages. Rows of Table 4 correspond to variables of the four-item assessment scale from the questionnaire (1 – not significant; 2 – somewhat significant; 3 – significant; 4 – extremely significant).

Table 5 contains upper and lower percentages of assessments, calculated using a 95% confidence interval, computed through bootstrap resampling with 1000 replications. For example, the percentage of score 3, calculated with a 95% confidence interval, is between 40 and 80.

Table 4. Architectural practice: Identification of key system stakeholders (Source: Authors)

		Frequency	Percent	Valid Percent	Cumulative Percent	Bootstrap for Percent	
						Bias	Std. Error
Valid	1.00	2	10.0	10.0	10.0	.0	6.6
	2.00	3	15.0	15.0	25.0	.1	8.4
	3.00	12	60.0	60.0	85.0	.2	11.1
	4.00	3	15.0	15.0	100.0	-.3	7.8
	Total	20	100.0	100.0		.0	.0

Table 5. Architectural practice: Identification of key system stakeholders (Source: Authors)

		Bootstrap for Percent	
		95% Confidence Interval	
		Lower	Upper
Valid	1.00	.0	25.0
	2.00	.0	35.0
	3.00	40.0	80.0
	4.00	.0	30.0
	Total	100.0	100.0

Based on results obtained in this fashion, summary tables of dichotomized data were made for each architectural practice, showing their significance to agile development processes (Table 6). Significance of explicit architectural practices is measured as the proportion of respondents that rated them as significant. Value of the indicator is calculated using Formula (1), which involves adding values of items 3 and 4 from the Percent column of Table 4, which correspond to “significant” and “extremely significant” assessments in the questionnaire.

$$\text{Proportion of respondents who rated a practice as significant} = \\ = (\text{value of Percent [row 3]} + \text{value of Percent [row 4]}) / 100. \quad (1)$$

In the example of the chosen architectural practice (Table 4), the proportion of respondents that rated the given architectural practice as significant was calculated by adding percent value of item 3, which was 60%, and percent value of item 4, which was 15%. The total of 75% was subsequently divided by 100 to calculate the proportion of respondents that rated the given architectural practice as significant.

Table 6 lists all identified architectural practices, ranked by their significance in the agile development process, according to the respondents. Based on the values of significance indicators, architectural practices were divided into 3 categories: highly significant, significant and insignificant explicit architectural practices to agile development processes. The presented results represent the answer to the third research question.

Architectural practices with significance indicator values between 0.8 and 1 were categorized as highly significant practices. Architectural practices with significance indicator values between 0.7 and 0.8 were categorized as significant, while those with significance indicator values less than 0.7 were categorized as insignificant.

Table 6. Significance of explicit architectural practices in agile development process, according to the respondents (Source: Authors)

Architectural practice	Proportion of respondents who rated a practice as significant
Forming a suitable team and choosing a software architect with regard to the problem being solved	Highly significant: 0.95
Understanding of the business problem	Highly significant: 0.95
Code review	Highly significant: 0.95
Active discussions with stakeholders aimed at analyzing and understanding the business	Highly significant: 0.9
Identification of architecturally significant requirements	Highly significant: 0.9
Risk analysis aimed at identifying and isolating areas of complexity	Highly significant: 0.9
Examining technology suitable for implementation	Highly significant: 0.9
Identification and definition of basic structures (modules) for the system core, as well as their relations (envision architecture)	Highly significant: 0.9
Testing system performance and other critical non-functional requirements	Highly significant: 0.9
Project scoping	Highly significant: 0.85
Analysis of dependencies between functional requirements and architectural elements during release planning	Highly significant: 0.85
Continuous architect's support throughout all key design issues	Highly significant: 0.85
Configuration management	Highly significant: 0.85

Creation of a common prioritized list of functional and non-functional requirements	Highly significant: 0.8
Definition of core data architecture	Highly significant: 0.8
Examination and development of a deployment model	Highly significant: 0.8
Validation of critical architectural requirements through prototyping	Highly significant: 0.8
Specification of integration tests	Highly significant: 0.8
Identification of key system stakeholders	Significant: 0.75
Formal architecture review	Significant: 0.75
Test case specification	Significant: 0.75
Release planning with a strategy focused on investigating legacy systems, dependencies on other partner/third party products, and data backward compatibility	Significant: 0.7
Acceptance test specification	Significant: 0.7
Development/assessment of QA tests	Significant: 0.7
Development of code-writing guidelines and other guidelines for system design	Insignificant: 0.6
Development of top-level documentation	Insignificant: 0.55
Managing dependencies with external systems that the system interacts with throughout a release	Insignificant: 0.5
Technical debt management, with focus on non-functional requirements in each iteration	Insignificant: 0.45
Defining detailed design of each module	Insignificant: 0.4
Detailed design documentation	Insignificant: 0.4
Examining and improving detailed design	Insignificant: 0.4

5. Threats to Validity

When it comes to external threats to validity, one would most certainly be that the study was limited to only one country. This was mitigated by including the respondents outsourced by companies from various countries, as well as ones employed by global software development organizations. Another threat is that no similar empirical research was carried out in Serbia before. This was mitigated by comparing the obtained results with the results of similar studies from around the world. Existing approaches and frameworks, such as LeSS [61], SAFe [62], or DAD [10] were not analyzed in this paper, despite their indubitably noteworthy impact on the agile community. The focus of this paper was on identifying explicit architectural activities, while future research is planned to result in a framework for their integration into agile development processes. Future research will also encompass an analysis of all existing frameworks for scaling up agile processes, as well as a comparison of these frameworks with the framework to be developed by the authors.

The authors aimed to contain the internal threats to validity by exercising methodological strictness, aimed at ensuring reliability and validity of the conclusions. The fact that panel was composed of respondents with expert knowledge and interest in the research problem, in line with the recommendations by Goodman [63], contributed to the validity of the research results, as well as insistence that all reposndents choices be argumented and detailed. The Content Validity Index technique, computed in accordance with the procedure prescribed by Polit and Beck [25], was used to ensure the validity of the research instruments developed for this study.

The reliability was increased by interviewing the respondents separately, which was aimed at eliminating the effect of group bias and conformity. Greater reliability of research results was ensured by bootstrapping with 1000 replications.

Criteria proposed by Lincoln and Guba [64] were used to ensure the credibility of results and conclusions of the qualitative part of the research. The criteria include: Applicability (Transferability), Consistency (Dependability), Neutrality (Confirmability), and Credibility.

The following set of techniques was used for answering the questions listed above [64]:

1. The technique used for ensuring applicability of the findings involved detailed description of the studied phenomenon, so that an external evaluator would be able to asses to which extent the conclusions are applicable to different situations, times, settings, or people. This ensured the basis for external validity of research findings.
2. The technique used for ensuring the consistency of the findings involved external review by two researchers, both PhDs. The aim was to assess the validity and whether the findings, their interpretations and conclusions were derived correctly from the data. This warranted greater accuracy and validity of the research and the findings.
3. The technique used for ensuring the neutrality of findings involved a detailed description of all steps of the research process. The descriptions contain all information on the course of the research (development of research instruments, field notes, raw data, etc.), as well as all activities that were carried out (qualitative data analysis, synthesis, identified topics, concepts, categories, etc).
4. The technique used for ensuring the credibility of findings involved setting aside a portion of empirical data and excluding it from the first iteration of the analysis. The analysis was initially carried out on remaining data, which produced the preliminary results. The stored data was subsequently analyzed in order to validate the conclusions.

6. Related Work and Discussion

Traditional development processes, which once successfully fulfilled business requirements and needs in the development of complex software solutions, can no longer adequately respond to new business challenges that modern organizations are faced with. This is the reason why agile processes are gaining popularity in the development of complex systems. The greatest challenge in the development of complex systems using agile processes is to develop strong architecture, while preserving the agility of the development process. Emergent architecture did not prove

to be sufficient for developing strong architecture of complex systems. Hence, agile processes need to be scaled up by incorporation of significant explicit architectural practices.

Unlike in traditional development, where architectural activities are concentrated in the initial phases of the project, in agile processes, architectural issues are addressed throughout the entire development lifecycle. However, research results show that surveyed agile practitioners rated explicit architectural practices from the initial phases of the traditional development (planning and scoping) as the most significant ones.

Even though it opposes the Agile Manifesto, which advocates "responding to change over following a plan" [60], as well as the XP mantra "YAGNI" (You Ain't Gonna Need It) [12], it is clear that traditional architectural practices are often applied in agile development processes. Efficient management of such software development projects necessitates a systematic approach to establishing a balance between traditional and agile architectural strategies. Furthermore, it is evident that agile architects highly value activities such as testing and code review. Such findings are in line with results that Hadar and Sherman [49] obtained in their empirical research.

Results also suggest that agile practitioners have a disinclination to documenting and detailed design, which supports the claim that they are attempting to find an optimal balance between agile and traditional approaches. As far as detailed design planning is concerned, agile practitioners do not consider it to be a significant architectural activity, but view it as a programmers' responsibility. The respondents do not consider source code to be the only necessary form of documentation in the complex system development projects, which is in line with the findings of Coplien and Bjørnvig [65]. Documenting activities are reduced to a minimum, and are mostly carried out by use of wikis, without some formal template or a structure. Architecturally significant requirements, decision on the technological stack, and architectural models in the form of a flowchart are most frequently documented. There is an apparent problem among practitioners due to the fact that there is no documentation of the reasons and rationale for the major portion of architectural decisions.

Even though agile processes do not formally recognize the role of a software architect, research suggests that such role exists in agile teams. However, it differs from the traditional one. The principal difference lays in software architects' continuous engagement throughout the whole development process. The respondents even stated that software architects often participate in the implementation of the solution, which points to the fact that agile teams employ a practice close to the organizational pattern 'Architect also Implement' [66]. Such findings are consistent with previous experiences, which emphasize how the classic role of a software architect is changed in agile processes: Hadar and Sherman [49] highlights that it is necessary to include the software architect in the whole agile development process, while Blair et al. [44] point to the need for architect's close collaboration with the project team and continuous exchange of ideas throughout the whole project. Furthermore, Hopkins and Harcombe [50] claim that the software architect's role is indispensable in the development of large-scale, complex systems. Faber [37] believes that architects need to be service providers to both programmers and clients, assuming different roles in interactions with both groups, while Madison [51] claims that a software architect should bind the agile process with software architecture development methods. Therefore, they must have a solid understanding of the agile process, establish a balance between business and architectural priorities, as to develop an agile architecture and utilize the benefits of both

approaches. It is safe to say that, as the complexity level rises, the agile concept of an omniscient "super engineer" becomes deficient, while a software architect is invited to compensate their lack of knowledge.

Research results clearly suggest that agile teams feel the need to scale up agile processes. This is evidenced by a set of explicit architectural practices typical to traditional development, rated by experts as highly significant in the agile development of complex software solutions. Furthermore, it can be concluded that the set of architectural practices rated as significant and highly significant is aligned with the practical architectural issues identified and classified in the paper. This substantiates the claim that agile teams are aware that these issues need to be resolved, which indicates a change in their attitudes towards this problem. In conclusion to his research, Ambler [67] stated agile teams' unawareness of the need for modifying agile development strategy to suit the complexity of the domain as the reason for scarce application of agile processes in the development of complex software solutions. He further explained that techniques and practices of agile processes, proven to be successful in projects from simple domains, do not guarantee success in the development of complex solutions, but rather need to be scaled up. Research behind this paper presents the evolution of agile teams' attitudes from those recognized in 2009. Results of the research carried on practitioners suggest that agile teams consider the domain to be an important factor in defining the architectural strategy. To be precise, the complexity of the domain, as well as team members' domain knowledge fall into the factors that determine how many up front architectural decisions will need to be made on the project, and how much time the team will need to spend on the architectural analysis of the system. This result is in accordance with the debate between Coplien and Martin [68].

7. Concluding Remarks

Empirical results of the research show that agile teams are attempting to reach their own solutions for modifying agile processes. All of the 20 respondents stated that they use a modified agile development process, with the intent to keep the scope flexible, so that it can be changed during the development process, should changes prove to be necessary.

The challenge of establishing balance between agile processes and traditional architectural practices in the development of complex software solutions requires joint efforts by practitioners and researchers. Although the interest in this topic has grown over the recent years, it can still be concluded that research papers restated to this topic based on empirical findings are still scarce.

Agile practitioners have not only recognized the need for incorporating explicit architectural practices into agile development processes, but also pointed to explicit architectural practices they consider suitable to be incorporated.

It is evident that agile processes are adopting more and more elements of the traditional development, hence the term 'traditionalisation'. The principal values of agile development established in the Agile Manifesto are still valued as highly as ever. However, in terms of agile development of complex systems, the traits of traditional methods, such as structured processes, use of specific tools, documentation, and plan driven development, are gaining recognition. The results presented in this paper encourage further efforts on finding the solution for integration of explicit architectural

practices in agile development, but making sure that the values that distinguish the agile processes remain fully preserved.

Directions for future research, aimed at facilitating more extensive application of agile processes in the development of complex software solutions, are set in accordance with the conclusions given above. Further actions will be focused on determining the critical points in agile processes that need to incorporate significant explicit architectural practices identified in this research. The ultimate goal of future research is to develop a framework for incorporation of explicit architectural practices into critical points of the agile process. In this way, agile teams would be offered clear directions on how to adapt their processes for the development of complex software solutions. Future research will also involve an analysis of all existing frameworks for scaling up agile processes (such as SAFe, LeSS, and DAD), and comparison of these frameworks with the framework to be developed by the authors of this paper.

References

1. Vranić, V.: Promoting Natural Human Attitude Towards Work: Scrum. In Proceedings of IV naučni skup Mreža. Singidunum University, Valjevo, Serbia, 8–12. (2013)
2. Imache, R., Izza, S., Ahmed-Nacer, M.: An Enterprise Information System Agility Assessment Model. Computer Science and Information Systems, Vol 9, No. 1, 107–133. (2012)
3. McDermid, J. A.: Complexity: Concept, Causes and Control. In Proceedings of the Sixth IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2000). IEEE Computer Society, Tokyo, Japan, 2–11. (2000)
4. Meadows, D. H.: Thinking in Systems: A Primer White River Junction. Chelsea Green Publishing, USA (2008)
5. Kruchten, P.: Scaling down projects to meet the Agile sweet spot. (2004). [Online]. https://people.eecs.ku.edu/~saiedian/811/Lectures/Misc/Papers-Agility/large-agile-projects.pdf (current December 2016)
6. Kruchten, P.: Contextualizing Agile Software Development. Journal of Software Evolution and Process, Vol 25, No. 4, 351-361. doi:10.1002/smr.572 (2013)
7. Matkovic, P., Tumbas, P., Sakal, M.: The RSX model: traditionalisation of agility. Strategic Management, Vol 16, No. 2, 74–83. (2011)
8. Nord, R. L., Tomayko, J. E.: Software architecture-centric methods and agile development. IEEE Software, Vol. 23. doi: 10.1109/MS.2006.54. (2006)
9. Kruchten, P.: Voyage in the agile memeplex. ACM Queue, Vol 5, No. 5, 38–44. (2007)
10. Ambler, S. W., Lines, M.: Disciplined agile delivery (1st ed.). IBM Press, Boston, MA, USA. (2013)
11. Babar, M. A.: Making Software Architecture and Agile Approaches Work Together. In M. A. Babar, A. W. Brown, I. Mistrik (Eds.), Agile software architecture (1st ed.). Elsevier, Waltham, MA, USA, 43–76. (2014)
12. Beck, K.: Extreme Programming Explained: Embrace Change (2nd ed.). Addison-Wesley, Boston, MA, USA. (2004)
13. Thapparambil, P.: Agile architecture: pattern or oxymoron? Agile Times, Vol 6, No. 1, 43–48. (2005)
14. Parsons, R.: Architecture and agile methodologies—how to get along. In Working IEEE/IFIP Conference on Software Architecture. WICSA, Vancouver, Canada. (2008)

15. 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, 1–16. (2005)
16. 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. Limerick, Ireland, 238–243. (2008)
17. Stal, M.: Refactoring Software Architectures. In Agile Software Architecture: Aligning Agile Processes and Software Architectures. Elsevier, Waltham, MA, USA, 130–152. (2014)
18. Babar, M. A.: An Exploratory Study of Architectural Practices and Challenges in Using Agile Software Development Approaches. In Software Architecture, 2009 European Conference on Software Architecture. Joint Working IEEE/IFIP Conference. IEEE, Cambridge, United Kingdom, 81–90. (2009)
19. Babar, M. A., Ihme, T., Pikkarainen, M.: An Industrial Case of Exploiting Product Line Architectures in Agile Software Development. In 13th international conference on software product lines (SPLC) (pp. 171–179). CMU, Pittsburgh, PA, USA (2009)
20. Abrahamsson, P., Babar, M. A., Kruchten, P.: Agility and Architecture: Can They Coexist? IEEE Software, Vol 27, No. 2, 16–22. (2010)
21. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. MIS Quarterly, Vol 28, No. 1, 75–105. (2004)
22. Kitchenham, B.: Procedures for performing systematic reviews. (2004). [Online]. Available: http://people.ucalgary.ca/~medlibr/kitchenham_2004.pdf (current August 2015)
23. Miles, M. B., Huberman, A. M.: Qualitative data analysis: An expanded sourcebook (2nd ed.). Sage. Los Angeles, CA, USA. (1994)
24. Dyba, T., Dingsoyr, T.: Empirical studies of agile software development: A systematic review. Information and Software Technology, Vol. 50, 833-859. doi: 10.1016/j.infsof.2008.01.006. (2008)
25. Polit, D. F., Beck, C. T.: The Content Validity Index: Are You Sure You Know What's Being Reported? Critique and Recommendations. Research in Nursing and Health, Vol. 29, No. 5, 489–497. doi: 10.1002/nur.20147. (2006)
26. Skulmoski, G. J., Hartman, F. T., Krahn, J.: The Delphi method for graduate research. Journal of Information Technology Education, Vol 6, No. 1, 1-21. (2007)
27. Ziglio, E.: The Delphi method and its contribution to decision-making. In E. Adler,M., Ziglio (Ed.), Gazing Into the Oracle: the Delphi Method and Its Application to Social Policy and Public Health (1st ed.). Jessica Kingsley Publishers, London, UK, 3–33. (1996)
28. Manning, C. D., Schütze, H.: Foundations of Statistical Natural Language Processing. The MIT Press, Cambridge, Massachusetts, London, UK. (1999)
29. Taylor-Powell, E., Renner, M.: Analyzing Qualitative Data. (2003). [Online]. Available: http://learningstore.uwex.edu/assets/pdfs/g3658-12.pdf (current November 2015)
30. Zhang, Y., Wildemuth, B. M.: Qualitative analysis of content. In Applications of Social Research Methods to Questions in Information and Library Science (1st ed.). Libraries Unlimited, Westport, CT, USA, 308–319. (2009)
31. Ristić, Ž.: Kvantitativna, kvalitativna i mešovita istraživanja, metodološki aspekti. Univerzitet u Novom Sadu, Novi Sad, Serbia. (2011)
32. Friedrichsen, U.: Opportunities, Threats, and Limitations of Emergent Architecture. In I. Babar, M.A., Brown, A. W., Mistrik (Ed.), Agile Software Architecture Aligning Agile Processes and Software Architectures (1st ed.). Elsevier, Waltham, MA, USA, 335-355. (2014)
33. Bellomo, S., Nord, R. L., Ozkaya, I.: A Study of Enabling Factors for Rapid Fielding Combined Practices to Balance Speed and Stability. In 35th International Conference on Software Engineering (ICSE). IEEE, New York, NY, USA, 982–991. (2013)
34. Cleland-Huang, J., Czauderna, A., Mirakhorli, M.: Driving Architectural Design and Preservation from a Persona Perspective in Agile Projects. In M. A. Babar, A. W. Brown, I.

Mistrik (Eds.), Agile Software Architecture Aligning Agile Processes and Software Architectures (1st ed.). Elsevier, Waltham, MA, USA, 83-111. (2014)
35. Jeon, S., Han, M., Lee, E., Lee, K.: Quality Attribute Driven Agile Development. In 9th ACIS International Conference on Software Engineering Research, Management and Applications. IEEE Computer Society, Baltimore, MD, USA, 203–210. doi: 10.1109/SERA.2011.24. (2011)
36. Brown, N., Nord, R., Ozkaya, I.: Enabling Agility Through Architecture. CrossTalk, Vol 23, No. 6, 12–16. (2010). [Online].
Available: http://static1.1.sqspcdn.com/static/f/702523/9627808/1291147379607/201011-0-Issue.pdf?token=PWhMQ%2FO3zkIAMF4Vzb7VQ8Q%2FeaE%3D (current September 2014)
37. Faber, R.: Architects as Service Providers. IEEE Software, Vol 27, No. 2, 33–40. doi: 10.1109/MS.2010.25. (2010)
38. Isotta-Riches, B., Randell, J.: Architecture as a Key Driver for Agile Success: Experiences At Aviva UK. In M. A. Babar, A. W. Brown, I. Mistrik (Eds.), Agile Software Architecture: Aligning Agile Processes and Software Architectures (1st ed.). Elsevier, Waltham, MA, USA, 357–374. (2014)
39. Weitzel, B., Rost, D., Scheffe, M.: Sustaining Agility through Architecture: Experiences from a Joint Research and Development Laboratory. In 2014 IEEE/IFIP Conference on Software Architecture (WICSA). IEEE Computer Society, Sydney, Australia, 53-56. doi: 10.1109/WICSA.2014.38. (2014)
40. Nord, R. L., Ozkaya, I., Sangwan, R. S.: Making Architecture Visible to Improve Flow Management in Lean Software Development. IEEE Software, Vol 29, No. 5, 33–39. doi: 10.1109/MS.2012.109. (2012)
41. Boehm, B., Lane, J., Koolmanojwong, S., Turner, R.: Architected Agile Solutions for Software- Reliant Systems. In T. Dingsøyr, T. Dybå, N. B. Moe (Eds.), Agile SoftwareDevelopment: Current Research and Future Directions. Springer, Heidelberg, Germany, 165–184. doi: 10.1007/978-3-642-12575-1. (2010)
42. Waterman, M., Noble, J., Allan, G.: How much architecture? Reducing the up-front effort. In AGILE India, 2012. IEEE, Le Meridien, Bengaluru, India, 56–59. doi: 10.1109/AgileIndia.2012.11. (2012)
43. Buchgeher, G., Weinreich, R.: Continuous Software Architecture Analysis. In M. A. Babar, A. W. Brown, I. Mistrik (Eds.), Agile Software Architecture: Aligning Agile Processes and Software Architectures (1st ed.). Elsevier, Waltham, MA, USA, 161-188. (2014)
44. Blair, S., Watt, R., Cull, T.: Responsibility-Driven Architecture. IEEE Software, Vol 27, No. 2, 26–32. doi: 10.1109/MS.2010.29. (2010)
45. Ven, J. S. van der, Bosch, J.: Architecture Decisions: Who, How, and When? In M. A. Babar, A. W. Brown, I. Mistrik (Eds.), Agile Software Architecture. Aligning Agile Processes and Software Architectures. Elsevier, Waltham, MA, USA, 113-136. (2014)
46. Kruchten, P.: Software architecture and agile software development: a clash of two cultures? In 2010 ACM/IEEE 32nd International Conference on Software Engineering. IEEE, Cape Town, South Africa, 497–498. doi: 10.1145/1810295.1810448. (2010)
47. Chen, L., Babar, M. A.: Towards an Evidence-Based Understanding of Emergence of Architecture Through Continuous Refactoring in Agile Software Development. In Software Architecture (WICSA), 2014 IEEE/IFIP. IEEE, Sydney, Australia, 195-204. doi: 10.1109/WICSA.2014.45. (2014)
48. Hadar, E., Silberman, G. M.: Agile Architecture Methodology: Long Term Strategy Interleaved with Short Term Tactics. In OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. ACM, New York, NY, USA, 641-651. doi: 10.1145/1449814.1449816. (2008)
49. Hadar, I., Sherman, S.: Agile vs. plan-driven perceptions of software architecture. In Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International

Workshop on. IEEE, Zürich, Switzerland, 50–55. doi: 10.1109/CHASE.2012.6223022. (2012)
50. Hopkins, R., Harcombe, S.: Agile Architecting: Enabling the Delivery of Complex Agile Systems Development Projects. In M. A. Babar, A. W. Brown, I. Mistrik (Eds.), Agile Software Architecture: Aligning Agile Processes and Software Architectures (1st ed.). Elsevier, Waltham, MA, USA, 291-314. (2014)
51. Madison, J.: Agile Architecture Interactions. IEEE Software, Vol 27, No. 2, 41–48. doi: 10.1109/MS.2010.35. (2010)
52. Qureshi, M. R. J.: Agile software development methodology for medium and large projects. IET Software, 6(4), 358–363. doi: 10.1049/iet-sen.2011.0110. (2012)
53. Hinsman, C., Sangal, N., Stafford, J.: Achieving Agility through Architecture Visibility. In QoSA '09: Proceedings of the 5th International Conference on the Quality of Software Architectures: Architectures for Adaptive Software Systems. Springer-Verlag, Berlin Heidelberg, Germany, 116–129. doi: 10.1007/978-3-642-02351-4_8. (2009)
54. Keuler, T., Wagner, S., Winkler, B.: Architecture-aware Programming in Agile Environments. In 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture. IEEE, Helsinki, Finland, 229–233. doi: 10.1109/WICSA-ECSA.212.35. (2012)
55. Hadar, I., Sherman, S., Hadar, E., Harrison, J. J.: Less is more: Architecture documentation for agile development. In Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop. IEEE, San Francisco, CA, USA, 121-124. doi: 10.1109/CHASE.2013.6614746. (2013)
56. Pareto, L., Sandberg, A., Eriksson, P., Ehnebom, S.: Prioritizing Architectural Concerns. In Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on. IEEE, Boulder, CO, USA, 22-31. doi: 10.1109/WICSA.2011.13. (2011)
57. Falessi, D., Cantone, G., Sarcia', S. A., Calavaro, G., Subiaco, P., D'Amore, C.: Peaceful Coexistence: Agile Developer Perspectives on Software Architecture. IEEE Software, Vol 27, No. 2, 23–25. doi: 10.1109/MS.2010.49. (2010)
58. Tyree, J., Akerman, A.: Architecture decisions: demystifying architecture. IEEE Software, Vol 22, No. 2, 19-27. doi: 10.1109/MS.2005.27. (2005)
59. Eloranta, V.-P., Koskimies, K.: Aligning Architecture Knowledge Management with Scrum. In Proceedings of the WICSA/ECSA 2012. ACM, Helsinki, Finland, 112–115. doi: 10.1145/2361999.2362023. (2012)
60. Beck, K., Beedle, M., Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, J.: The Manifesto for Agile Software Development. (2001). [Online]. Available: http://agilemanifesto.org/. (current September 2014)
61. LeSS Company B.V.: LeSS Framework. (2014-2017). [Online]. Available: https://less.works/less/framework/index.html (current July 2017)
62. Scaled Agile, Inc.: Scaled Agile Framework – SAFe for Lean Enterprises. (2010-2017). [Online]. Available: http://www.scaledagileframework.com/ (current July 2017)
63. Goodman, C. M.: The Delphi technique: a critique. Journal of Advanced Nursing, Vol 12, No. 6, 729–734. (1987).
64. Lincoln, Y. S., Guba, E. G.: Naturalistic Inquiry. Sage Publications, London, UK. (1985)
65. Coplien, J. O., Bjørnvig, G.: Lean Architecture: for Agile Software Development. Wiley, Chichester, UK. (2010)
66. Frťala, T., Vranić, V.: Animating Organizational Patterns. In 2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering. IEEE. USA, 8–14. doi: 10.1109/CHASE.2015.8 (2015)
67. Ambler, S. W.: Agility at Scale Survey Results from the November 2009 DDJ State of the IT Union Survey. Surveys Exploring The Current State of Information Technology Practices. (2009). [Online]. Available: http://www.ambysoft.com/surveys/ (current September 2014)

68. Marinescu, F.: Coplien and Martin Debate TDD, CDD and Professionalism (2008). [Online]. https://www.infoq.com/interviews/coplien-martin-tdd (current December 2016)

Predrag Matković is an assistant professor at the University of Novi Sad, Faculty of Economics in Subotica, lecturing in the field of business informatics on undergraduate, master, and PhD studies. He authored numerous journal and conference papers, and has been a member of several national and international research project teams. His research interests include: business process management, agile software development, object-oriented analysis and design, and integrated ERP, SCM, CRM software solutions.

Mirjana Marić is a teaching assistant at the University of Novi Sad, Faculty of Economics in Subotica, lecturing in the field of business informatics, with special research interests in analysis and design of information systems, agile software development and integrated ERP, SCM, CRM software solutions. She is the author and co-author of numerous journal and conference papers, and she participated on several national and international research projects.

Pere Tumbas is full professor at the University of Novi Sad, Faculty of Economics in Subotica in the field of Business Informatics. He authored and co-authored several textbooks, handbooks and monographs, as well as over 250 works presented and published at international and national conferences, symposia and journals in the field of Business Informatics. During more than 35 years of working at the Faculty, he has participated in a number of scientific and professional projects and studies. He is an editor or editorial board member of in several international journals, as well as a member of several program committees of scientific conferences. His primary areas of interest are: business analysis, business modeling, business process management, management information systems, digitalization, and business transformation.

Marton Sakal is an associate professor at the Faculty of Economics Subotica, University of Novi Sad. He received his PhD in Business Informatics, and currently teaches courses in programming, enterprise application development, and business information systems. Professional interests: information system development, user-centered interface design, end-user development, web 2.0, web programming. Authored and co-authored about 100 papers, published in journals or presented at national and international conferences, co-authored a workbook on programming.

Received: August 20, 2016; Accepted: July 20, 2017.