Combining Agile and Traditional Methodologies in Medical Information Systems Development Process

PETAR RAJKOVIC, University of Nis, Faculty of Electronic Engineering
IVAN PETKOVIC, University of Nis, Faculty of Electronic Engineering
ALEKSANDAR MILENKOVIC, University of Nis, Faculty of Electronic Engineering
DRAGAN JANKOVIC, University of Nis, Faculty of Electronic Engineering

Having experience with different traditional and agile methodologies in medical information system development and upgrade projects, we support the statement that no single methodology can be used with its full potential in all cases. Instead, mentioned methodologies have their own place in complex software projects and within certain category of sub-projects give the best results. In this paper we presented the guideline on choosing adequate methodology in different development and upgrade projects. All examined projects are defined within the scope of larger medical information system called Medis.NET developed by our research group, and deployed in 30 healthcare institutions. The bottom-line is that during requirement collection we rely on model and feature driven development supported by automatic code generation tools, which lead to faster prototype generation. Next, dividing the set of functionalities to sub-projects we group them in three categories – core projects, user-oriented, and loosely coupled application. For core projects we find traditional methodologies suitable since their outcome is a stable process and rich documentation. User-oriented sub-projects are developed under standard SCRUM, while for loosely coupled applications we chose Lean. We measured the effects of chosen approaches comparing time spent against estimated time for development, and by analyzing code metrics. Code metrics were automatically generated by our development environment, Microsoft Visual Studio, and we take into account few major values such are maintainability index, lines of codes, class coupling and cyclomatic complexity. In addition, we propose an adaptation for Lean model which gave us promising results in software upgrade projects. The proposed approach put additional responsibility to lead architects, since have to properly identifies all pros and cons for different methodologies and, knowing this well, wisely choose the right methodology for the right sub-project.

Categories and Subject Descriptors: **H.4.0 [Information Systems]**: General; **D.2.8 [Software Engineering]**[1]**:** Metrics; **D.2.2 [Software Engineering]:** Design Tools and Techniques

General Terms: Design, Management

Additional Key Words and Phrases: Traditional software development, Agile software development, Lean, SCRUM

1. INTRODUCTION AND MOTIVATION

Since 2002 our research group was involved in many development and upgrade projects targeting medical information systems (MIS) [Rajkovic et al. 2005, 2013]. Initially, we relied on standardized traditional methodologies such are Waterfall, Spiral [Boehm 1988] and Rational Unified Process (RUP) [Kruchten 2004]. The main advantage of traditional methodologies was, and still is, a concise set of guidelines and significant volume of produced documentation [Leau 2012]. The main problem were long phases resulting with late response to potential changes requested by the end user. By introducing agile methodologies [Beck et al. 2001], primarily SCRUM [Schwaber et al. 2002], that require more interaction with the end users, we managed to reduce the number of new, surprising requests coming in the later stages of the project [Black et al. 2009]. On the other side, we faced with the set of new and unexpected problems – problems with lower volume of generated documentation as

This work is supported by the Ministry of Education and Science of Republic of Serbia (Project number #III47003).
Authors' addresses: Petar Rajkovic, University of Nis, Faculty of Electronic Engineering – Lab 534, Aleksandra Medvedeva 14, 18000 Nis, Serbia, e-mail: petar.rajkovic@elfak.ni.ac.rs; Ivan Petkovic, University of Nis, Faculty of Electronic Engineering – Lab 523, Aleksandra Medvedeva 14, 18000 Nis, Serbia, e-mail: ivan.petkovic@elfak.ni.ac.rs; Aleksandar Milenkovic, University of Nis, Faculty of Electronic Engineering – Lab 533, Aleksandra Medvedeva 14, 18000 Nis, Serbia, e-mail: aleksandar.milenkovic@elfak.ni.ac.rs; Dragan Jankovic, University of Nis, Faculty of Electronic Engineering – Lab 105, Aleksandra Medvedeva 14, 18000 Nis, Serbia, e-mail: dragan.jankovic@elfak.ni.ac.rs;

In: Z. Budimac, Z. Horváth, T. Kozsik (eds.): Proceedings of the SQAMIA 2016: 5th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Budapest, Hungary, 29.-31.08.2016. Also published online by CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073)

well as problems with social relations [Leau 2012]. For some users we become annoying and tedious, and some of our team members feels uncomfortable due to the frequent interaction with the clients.

With standard methodologies we realize that the stress level increases when project ends the development and has to enter to testing phase. With agile, this problem was also present, but stress-full intervals are scattered during the whole process and last shorter. After SCRUM, we looked for Lean methodology [Poppendieck 2003]. The main advantage was the task-based organization of development process. Having larger number of smaller tasks, allowed us to group them in logical packages and move these packages through Kanban table with different speed. This resulted in continuous system integration and our users were not so flooded with demo and test sessions.

When our first research and development project was about to finish in 2007, we had several pieces of software already finished and had the experience with few different software development methodologies (SDM). Based on that experience and lessons learned, we actually stop looking for a single methodology that fits all the cases of development and upgrade, but define the set of guidelines that would help us in choosing the adequate methodology for each sub-project. In 2008, we started with the preparation for new development project which primary output was a MIS dedicated for Serbian primary and ambulatory care. Defined set of the guidelines for choosing appropriate methodology we applied in mentioned development [Rajkovic et al. 2009]. Complete project consists of 33 development and upgrade subprojects run under different SDMs.

In related work section we described the most important works that were a great support to our idea. Next section this paper presents our guideline for choosing proper methodology for proper sub-project. Since we were encouraged with initial results, we decided to propose a modification of Lean approach by incorporating some SCRUM concepts, which is described in separate section. In result section, we evaluated our development and upgrade projects that was carried under Medis.NET software project. The evaluation is based on the code metrics provided by Microsoft Visual Studio and project timeline analysis.

2. RELATED WORK

The work presented in [Leau et al. 2012] clearly describes all the advantages and disadvantages for traditional and agile SDMs, claiming that adoption of proper methodology is crucial for governing software life cycle and later system acceptance. We faced with all of situations described in [Leau et al. 2012] in our projects developed in period 2002-2007, and start going to the idea that should not look to one almighty methodology, but to identify different types of projects and the methodologies which would lead to the most successful scenario. Due to frequent changes in many areas of stakeholder requests, as well as in legislative, in period of almost 15 years, we had enough opportunities to check different methodologies.

Using metrics to evaluate software projects maintainability is not a new idea. Since well-known work of [Coleman et al. 1994], many different studies were conducted. In that branch, we found [Doherty 2012] as important for our research. This paper presents analysis on the most important success factors for traditional and agile methodologies. This analysis shows that not methodologies themselves, but related factors such are user interaction level, team competence, or quality of collected requirements have bigger effect on development process organization and later system acceptance. In that light, it is easier to connect SDM with mentioned factors, and based on them chose a proper methodology.

Introducing hybrid development methodologies is also a topic discussed in literature. For example, [Runeson 2005] discusses introduction of agile methodologies to traditional stage gate model. According to this work, the role of gate model is in strategical planning, while agile approach suits better for “microplanning, day-to-day work control, and progress reporting”. Finding the suitable place for agile methodologies within large scale projects is described in [Lindvall et al. 2004]. Authors based the research on the data gathered from large companies (Daimler Chrysler, ABB, Nokia) and analyzed their experience. The general conclusion is that agile methodologies can match the needs for some specific types of the projects, while the major challenge lies in “efficient integration the agile project

into overall environment". During the last decade agile methodologies developed rapidly and latest researches conducted on overall use of SDMs show significant advantage of SCRUM versus Waterfall [Shane et al. 2015]. Unfortunately this research shows only overall statistics, but not drilled down to specific domains of developed projects.

Since we decided to use spiral model, SCRUM and Lean as base development methodologies, we rely on Model Driven Development (MDD) in requirement collection, and on Feature Driven Development (FDD) in prototype building phase. Works presented in [Torchiano et al. 2013] and [Loniewski et al. 2010] gave us well argumentation for such decision. Beside we tend to use MDD also in later stages, mentioned three methodologies are the ones that we followed in our projects. Another two important aspects for optimal methodology utilizations are planning [Virani et al. 2014] and customer communication [Korkala et al. 2010]. These two "environmental" variables should also be considered when it comes to choosing the methodology for specific project. In general, many sources advocating usage of mixed and/or blended methodologies can be found. For example, we rely on [Okada et al. 2014] and [Hoda et al. 2013] when defining the updated lean model for software upgrade projects.

3. GUIDELINES FOR CHOOSING A METHODOLOGY FOR DEVELOPMENT PROJECT

Basically, every SDM we used in our first development project (Waterfall/Spiral/RUP, SCRUM and Lean) has its own advantages and disadvantages [Black 2009]. For this reason, we defined a set of guidelines that should help us in defining cases when some SDM should be applied. The summary of the guidelines contains following statements:

- During requirement collection phase, it is necessary to start building domain models and clearly identify main features
- Identify extension points in created model as earlier as possible
- Use component generation tools in order to reduce development time
- Categorize sub-projects as core projects, user-oriented (frontend) and loosely coupled components
- For core projects use traditional or agile methodology familiar to all assigned team members
- Frontend project develop under SCRUM to ensure proper communication with end users
- For loosely coupled components use Lean
- When it comes to team members - take care not only on technical, but also social skills, and assign people accordingly
- If needed, experiment with new development approach only in low to medium scale fronted and loosely coupled
- Do not experiment with new development approach in mission critical or core components

We started applying it in year 2009 when development of Medis.NET began. Thanks to the earlier projects, we had enough domain and process knowledge to define basic entities for future system. Also, we had developed domain models as well as model based code generation tools that should help us in overall improve our development process. Mentioned MDD tools are intensively used during requirement specification and prototype building phase [Rajkovic et al. 2015]. Once, when requirement collection phase is finished and customer signs request document, we already have initial advantage, since, until that moment, our domain models are on the way to be finished, system architecture is defined and the most of sub-projects are identified and implementation technology is chosen.

Next step is to define sub-projects and to divide them in three major groups – front-end projects, back-end projects and loosely coupled components. As front end projects we categorize these that are rich with GUI elements and forms. The focus of future's everyday use, from the customer point of view, will be on them. For this reason, it is important to maintain constant contact with prospective users. These sub-projects are developed under SCRUM methodology. Not directly exposed to the customers are back-end related functionalities containing business logic and data processing functionalities. The projects belonging to this category make backbone of the system. Due to the importance, the most detailed documentation is required for this category of the projects. For this

reason, we usually choose some of traditional methodology based on a strong set of rules. Another advantage that traditional SDMs brought to this category of the projects are long testing and verification phases which are necessary for the most critical parts of the complete system [Black et al. 2009].

For partially independent and loosely coupled sub-systems, we choose Lean as a compromise between SCRUM and traditional methodologies. In MIS environment, to this category belong projects that are developed for some specific department usually requiring higher level of interaction with third party software and/or medical equipment. Next, common request is that these applications can work disconnected from the central MIS. The best examples for this type of the projects are applications for radiology and biochemical laboratory. Lean methodology looked as the good balance between SCRUM and traditional in this particular case. Equipment integration would be usually divided in sub-projects defined per communication protocol, which in reality means per device, or with larger institutions per devices that belongs in the same generation of certain manufacturer. In the scope of lean methodology all the tasks related to same communication protocol are grouped in the same work packages. Developed functionalities are then tested package by package, and the interaction with the users during the development process is reduced comparing to SCRUM.

4. LEAN MODEL ADAPTATION FOR SOFTWARE UPGRADE PROJECTS

The statuses that one task passes through according to the Lean model are: definition and moving to backlog, approving and moving to to-do list, designing, developing, testing and moving to the list of finished tasks. Initially, we fully relied on original lean approach. The good side of this approach was a possibility to group tasks in logical packages, and to change their statuses according to Kanban table. Grouping tasks to logical packages allowed us incremental development and partial delivery to the customer. On the other side, the main problems came in integration testing when discovered bugs can trigger more work needed than expected causing partial or even overall deadline extensions. Another weak spot, from our point of view, is moving tasks to to-do list. In the moment when task reaches to-do list, only general requests are known, and, often, followed with even more general assumptions. Due to this fact, design phase would then last longer, since design phase for upgrade projects need to include not only new functionalities, but also integration to existing system. This left a lot of space for misinterpretation and false assumptions. Even worse, this can happen also during development and testing phases. In upgrade projects, this kind of problems is common and even expected. This, it is important to identify them as earlier as possible and the react according the case.

For mentioned reasons we proposed the update of Lean approach for upgrade projects (Fig. 1). The proposed improvement is based on changing the role of design phase. Design phase will be split and partly incorporated into to-do list creation and partly to development phase. Looking at the design phase itself, we identified two major parts – functionality design and integration design. Designing integrational parts for the new components is proportional to their integration level. Thus, we considered this part of the designing phase is one of the most critical. In order to reduce the possibility for errors in later stages, this part of design phase is moved to to-do list creation.

The cost of this decision is slower forwarding tasks through Kanban table in initial phases, but the benefit is visible later when missing links are rarely identified. Another benefit we notice is an effective introduction of new team members to the functionality of the existing system. Working together with experience colleagues they are able to get the insight to the initial system and to better understand the role of the update they will work on. Additional set of tasks moved to this phase is the improvement of software's domain model (if needed) and the initial generation of the components that could be automatically generated. Now, when task is moved to to-do list, it has attached the document and code snippets needed for later development and the integration. Design phase now remains only with high-level design of new functionality followed with risk assessment. Detailed design is moved to development phase. Development phase is enriched also with initial testing that should bring verified component to the next stage of the process. Enriching the development phase by detailed planning on the beginning and by functional testing at the end, results in sub-projects consisting of several tasks

that can be treated as small software projects. These small software projects are then developed using SCRUM paradigm. We tend to keep these sub-projects as SCRUM guided projects with just few sprints that are responsible for development of integration ready components.

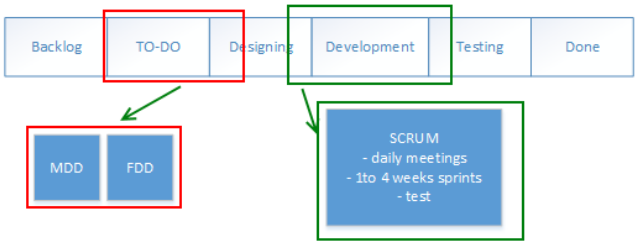

Fig. 1 Proposed Lean Model Adaptation

The next stage is testing. Since the functionality testing is done in development phase, the main part of testing phase is now integration testing. Since, in many cases functionality testing includes some segments of integration testing, this part is often used for internal acceptance test. On this way, the majority of bugs is identified earlier in development stage, so in ideal case, this step is more a formal verification. After this part is successfully finished, product is moved to the hands of prospective users for their acceptance testing. The main benefit of proposed update is lower number of moves back in the Kanban table, which results with faster development. Also, created code is with higher maintainability index (MI) easing future maintenance and further upgrades. In next chapter the results we get for projects developed under updated Lean are compared with these developed with standard Lean.

5. RESULTS AND DISCUSSION

With first MIS systems we developed, in period 2002-2006, we got the opportunity to test several SDM both for development and upgrade projects. As it has been explained in previous section, we decided to rely on MDD and FDD in early stages of development, but use traditional (Waterfall/Spiral), SCRUM and Lean for software development itself. [Atalag et al. 2014]. To calculate the effects of different methodologies we used standard code metrics offered by Microsoft Visual Studio development environment to check the code maintainability. To check development efficiency, we used ration between estimated and time needed for development. For calculating time wise efficiency, we used our projects logs and for code metrics calculation we used embedded code analysis tools that came as the standard part of Microsoft Visual Studio [Microsoft Library 2015]. As the leading parameter for the discussion, MI was used. Along with it, class coupling, number of lines of codes, cyclomatic complexity (CC), used methodology and project type are presented. We used our SDM assignment approach for total 33 sub-projects within our MIS (Table 1). CC is a measure of structural code complexity. Since it represents number of paths in the program, lower values are better. Higher CC results in harder code understanding and larger number of tests needed. CC should not be evaluated separately of the number of the lines of code. Higher number of lines of code will usually result in higher complexity number, so the relevant could be the ratio between complexity and lines of codes. As this value is lower, the code should be easier to maintain. At the end, the parameter named MI is a value in 0-100 range that should describe the relative easiness of code maintenance. Higher value is better. Microsoft states that minimal value claiming good maintainability is 20. Based on our experience, all the projects having this index lower than 50 are not quite easy maintainable especially for new team members. The formula for MI (taken from Microsoft Development Network) used in Visual Studio code analysis tool looks like:

Maintainability Index = MAX(0,(171 – 5.2 * ln(Halstead Volume) – 0.23 * (Cyclomatic Complexity) – 16.2 * ln(Lines of Code))*100 / 171)

Both best ratio between time used versus estimated (UvE) and the best overall MI we got with projects developed under Lean methodology. Between SCRUM and traditional methodologies,

traditional methodologies gave us slightly better values for development, while SCRUM was significantly better with upgrade projects. Looking the difference between development and upgrade project Lean and SCRUM gave better results with upgrade than development projects. On the other side, usage of standard methodologies in upgrade projects resulted with significantly lower MI.

Table 1 Medis.NET MIS – Comparison of the code quality for different projects (Columns: MI – maintainability index, LoC – lines of code, CC – cyclomatic complexity, RCC – relative cyclomatic complexity (CC/LoC), UM – used methodology, PT – project type, UvE – time used vs time estimated displayed as percentage)

Project Name	MI	LoC	CC	RCC	UM	PT	UvE
MedisControls	81	18131	4513	0.25	SCRUM	Dev	1.02
MedisData	86	19772	11115	0.56	Traditional	Dev	0.96
MedisDataAcquisitionLab	76	489	175	0.36	Lean	Upgrade	1.95
MedisDental	64	2293	582	0.25	SCRUM	Upgrade	0.88
MedisEHRCore	54	21571	2499	0.12	Traditional	Dev	0.91
MedisEHREditDialogs	61	25693	4422	0.17	SCRUM	Dev	1.12
MedisEHREditDialogsGin	57	1295	153	0.12	SCRUM	Dev	1.45
MedisEHREditDialogsIZIS	71	255	45	0.18	Modified Lean	Upgrade	0.75
MedisEHREditDialogsVisiting	44	10284	521	0.05	SCRUM	Upgrade	1.15
MedisEHREditDialogsSportMed	29	1266	30	0.02	Traditional	Upgrade	4.33
MedisEHREditDialogsStandards	62	1412	111	0.08	SCRUM	Upgrade	0.85
MedisEHREditDialogsDocuments	55	3706	435	0.12	Lean	Upgrade	1.04
MedisEHRFactory	66	286	67	0.23	Traditional	Dev	0.55
MedisFinanceReporting	77	10709	1967	0.18	Lean	Dev	0.92
MedisDoctorAssignment	77	1136	188	0.17	Lean	Dev	0.61
MedisIZISCore	89	1163	425	0.37	Modified Lean	Upgrade	0.80
MedisIZISExchange	83	658	276	0.42	Modified Lean	Upgrade	0.76
MedisCommunicationService	58	3989	641	0.16	SCRUM	Dev	1.01
MedisConfiguration	61	6454	838	0.13	Traditional	Dev	1.02
MedisLabIS	61	7506	1291	0.17	Lean	Dev	1.09
MedisOccupational	48	6042	324	0.05	Traditional	Upgrade	2.19
MedisTimeSchedule	67	3720	531	0.14	SCRUM	Upgrade	0.79
MedisDocumentsExchange	78	562	139	0.25	SCRUM	Upgrade	0.84
MedisFinance	42	1005	80	0.08	SCRUM	Dev	0.99
MedisReception	63	5600	951	0.17	SCRUM	Dev	1.06
MedisAccessControl	78	2693	941	0.35	Traditional	Dev	1.12
MedisRadiology	60	2328	298	0.13	Lean	Dev	0.97
MedisWorkOrganization	74	4853	956	0.20	Modified Lean	Upgrade	0.98
MedisReports	85	164	100	0.61	SCRUM	Dev	1.67
MedisSearchControls	66	4581	703	0.15	SCRUM	Upgrade	1.33
MedisInsuranceExchange	78	1312	467	0.36	Lean	Dev	2.01
MedisPatientScheduling	67	2538	528	0.21	SCRUM	Upgrade	0.89
MedisPatientDocumentation	62	4080	715	0.18	Lean	Dev	0.95

As it has been mentioned, we chose standard methodology twice for upgrade project. Once for sub-project for a department where (we believed) we had strong stakeholder request (MedisOccupational), while the another was the extension of EHR and corresponding GUI library for sports medicine department (MedisEHREditDialogsSportMed). With both cases we run into well-known problems with standard methodologies – we received too many update requests to late in project scope. Chasing the deadline, we ended up with on-site updates in fastest possible manner. Eventually, we get the projects with significantly lower MI. Luckily, project MedisEHREditDialogsSportMed having mentioned index of 29, has relatively small number of lines, so later bug fixes could be conducted without many problems. Problems with significant delays we experienced in projects where scope change and project extensions happened too often (such are reporting service and integration with

insurance fund). In this cases, chosen methodologies helped in a way that resulting code had acceptable MI, but the influence of selected methodology on overall delay was minor.

Table 2 Medis.NET MIS – code quality summary (Ovl – overall, Dev – development projects, Upg – upgrade projects)

	Number of projects			Maintainability index			Lines of code			Time needed vs time estimated		
Method	**Ovl**	**Dev**	**Upg**	**Ovl**	**Dev**	**Upg**	**Ovl**	**Dev**	**Upg**	**Ovl**	**Dev**	**Upg**
Lean	12	6	6	71.08	69.17	73.00	3183	4512	1854	1.00	1.02	0.96
Standard	8	6	2	68.25	69.17	65.50	3908	4512	2097	1.03	1.02	1.15
Modified	4	0	4	79.25	0.00	79.25	1732	0	1732	0.85	0.00	0.85
SCRUM	14	7	7	63.93	63.86	64.00	5922	7982	3627	1.07	1.06	1.08
Traditional	7	2	5	60.29	69.00	38.50	8297	10155	3654	1.15	0.95	2.56

Development projects that we run under traditional methodologies were core projects which specifications were designed even before our Medis.NET started. The design was the result of lessons learned from our initial set of projects, and during development only few inputs were added, updated or deleted. This helped us in calm development of core system resulting with well-documented and highly maintainable code. Nevertheless, the average volume of these projects is more than 10000 lines of code and CC is kept on satisfied level. SCRUM approach gave us similar average results both for development and upgrade projects. Beside the average volume of development project is more than twice than with upgrade, all parameters describing code maintainability is on similar rank. For SCRUM approach, we could just confirm our old finding that SCRUM works well when both sides (developer and users) are dedicated to get final goal.

We got the best result in projects where Lean approach was applied. Initially we tested modified Lean approach on the project which outcome should be time-scheduling component. Comparing with the results we got from the projects of similar volume, but developed with SCRUM and Lean, we get slightly higher maintainability, but project was finished much faster and with lower bug rate during acceptance phase. Updated Lean gave us very optimistic results, and with use on future projects having higher volume of code and functionalities, we expect to prove it as feasible. Currently, we are working on the project that should integrate different MIS systems used in Serbian public healthcare where updated Lean is applied.

6. CONCLUSION

This paper presents our heuristic approach, based on the experience of our research group, for choosing proper SDM for specific development and upgrade projects within MIS systems. After 15 years long experience with MIS development we tend to believe that there is (still) not a silver bullet when it comes to methodologies supporting software life cycle. Having experience with different traditional and agile SDMs we support the statement that all of the mentioned methodologies have their own place in complex software projects and within certain category of sub-projects give the best results. In this paper we presented our initial experience with traditional and agile models as well as the guideline how to choose adequate methodology in different MIS related development and upgrade projects. All examined projects are defined within the scope of larger MIS called Medis.NET developed by our research group. The bottom-line is that during requirement collection we rely on MDD and FDD approach supported by automatic code generation tools, which lead to faster prototype generation. Next, dividing the set of functionalities to sub-projects we group them in three categories – core projects, user-oriented, and loosely coupled application. For core projects we find that traditional SDM are suitable since their outcome should be stable process and rich documentation. User-oriented sub-projects are developed under standard SCRUM, while for loosely coupled applications we chose Lean.

We measured the effects of chosen approaches comparing time spent against estimated time for development, and by analyzing code metrics results. Code metrics were automatically generated by our development environment, and we take into account few major values such are MI, lines of codes, class coupling and CC. For development project we get satisfied results for each of three used

methodologies. Lower MI for projects developed under SCRUM is mostly the consequence of more frequent specification change due to more intense interaction with customers. For upgrade projects, SCRUM and Lean gave slightly better results, while the usage of standard methodologies in upgrade projects resulted with the significant overdue and extremely low MI in some cases. For upgrade projects we started specifying updated Lean approach based on combination of Lean and SCRUM. It is used in the latest upgrade projects and first results look promising. In all projects where it was used, the consequence was faster development and code with higher MI.

Next step in our research will be analysis of results from testing phase. In this paper, the focus was clearly on development, while the software verification, despite of its importance, was not covered in details. Extending presented results with testing, validation and verification analysis will help us in defining more detailed guidelines covering wider area of MIS life-cycle.

With this paper we wanted also to show that is important that software development process must be guided carefully, and that no methodology can be claimed as "the best for all purposes and in all cases". It is necessary that software architect properly identifies all pros and cons for different methodologies and, knowing this well, wisely choose the right methodology for the right project. Also, it is important to state that there is the place to combine different methodologies and lead the process with blended or mixed methodology. With proper methodology choosing guidelines and the introduction of even new promising approaches, development and upgrade processes for large and complex software projects, similar to Medis.NET, should be more effective, less stressful and gave more benefits both to developers and end users.

REFERENCES

Alexandra Okada, et al, 2014. Knowledge Cartography: software tools and mapping techniques. Springer

Barry W. Boehm. 1988. A spiral model of software development and enhancement. *Computer* 21.5 (1988): 61-72

Don Coleman et al. 1994. Using metrics to evaluate software system maintainability. *Computer* 27.8 (1994): 44-49.

Grzegorz Loniewski, Emilio Insfran, and Silvia Abrahão. 2010. A systematic review of the use of requirements engineering techniques in model-driven development. *Model driven engineering languages and systems*. Springer Berlin Heidelberg, 213-227.

Hastie Shane, Wojewoda Stephane, Standish Group 2015 Chaos Report. Infoq. https://www.infoq.com/articles/standish-chaos-2015

Ken Schwaber and Mike Beedle. 2002. Agilè Software Development with Scrum

Kent Beck et al. 2001. Manifesto for agile software development.

Koray Atalag et al. 2014. Evaluation of software maintainability with openEHR–a comparison of architectures. *International journal of medical informatics*, 83.11: 849-859.

Marco Torchiano et al. 2013. Relevance, benefits, and problems of software modelling and model driven techniques—A survey in the Italian industry. *Journal of Systems and Software* 86.8: 2110-2126

Mary Poppendieck and Tom Poppendieck. 2003. Lean Software Development: An Agile Toolkit: An Agile Toolkit. Addison-Wesley

Michael J. Doherty et al. 2012. Examining Project Manager Insights of Agile and Traditional Success Factors for Information Technology Projects: A Q-Methodology Study. *PhD, Marian University, Doctoral Dissertation*.

Microsoft Library. 2015. https://msdn.microsoft.com/en-us/library/bb385914.aspx

Mikael Lindvall et al. 2004. Agile software development in large organizations. *Computer*, 37.12: 26-34.

Mikko Korkala, Minna Pikkarainen, and Kieran Conboy. 2010. Combining agile and traditional: Customer communication in distributed environment. *Agility Across Time and Spac*e. Springer Berlin Heidelberg, 201-216.

Per Runeson et al. 2005. Combining agile methods with stage-gate project management. *IEEE software*, 3: 43-49.

Petar Rajković, Dragan Janković and Tatjana Stanković. 2009. An e-Health Solution for Ambulatory Facilities, *9th International Conference on Information Technology and Applications in Biomedicine*, ITAB 2009, Larnaca, Cyprus, November, 2009; ISSN: 978-1-4244-5379-5

Petar Rajković, et al. 2005. Cardio Clinic Information System Realization, *Electronics*, Vol. 9, No 1, pp, 41-45.

Petar Rajkovic, Ivan Petkovic, Dragan Jankovic. 2015. Benefits of Using Domain Model Code Generation Framework in Medical Information Systems. *Fourth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications SQAMIA 2015*. pp. 45-52

Philippe Kruchten. 2004. The rational unified process: an introduction. Addison-Wesley Professional.

Rashina Hoda, James Noble, and Simon Marshall. 2013. Self-organizing roles on agile software development teams. *Software Engineering, IEEE Transactions on* 39.3: 422-444.

Shamsnaz Virani and Lauren Stolzar. 2014. A Hybrid System Engineering Approach for Engineered Resilient Systems: Combining Traditional and Agile Techniques to Support Future System Growth. *Procedia Computer Science* 28: 363-369.

Sue Black et al. 2009. Formal versus agile: Survival of the fittest. *Computer* 42.9 : 37-45.

Yu Beng Leau et al. 2012. Software development life cycle AGILE vs traditional approaches. *International Conference on Information and Network Technology*. p. 162-167.