10. Piho, G., Tepandi, J., Parman M., Perkins, D.: From archetypes-based domain model of clinical laboratory to LIMS software. 33rd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatia, Croatia, May 24-28, accepted paper (2010)
11. Piho, G., Roost, M., Perkins, D., Tepandi, J.: Towards archetypes-based software development. In: International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CISSE 09), December 4 -12, accepted paper, (2009)
12. Piho, G.: Towards Archetypes and Archetype Patterns Based Software Engineering Techniques of Domains, Requirements and Software. In: Nordic workshop and doctoral symposium on dependability and security (NODES), Magnuse, Estonia, pp. 31-36 (2008)
13. Björner, D.: Domain Theory: Practice and Theories (A Discussion of Possible Research Topics). In: 4thInternational Colloquium on Theoretical Aspects of Computing - ICTAC, Macau SAR, China, (2007)
14. Björner, D.: Software Engineering, vol. 1: Abstraction and Modeling, Springer (2006)
15. Björner, D.: Software Engineering, vol. 2: Specifications of Systems and Languages, Springer (2006)
16. Björner, D.: Software Engineering, vol. 3: Domains, Requirements, and Software Design, Springer (2006)
17. Zachman, J. A.: A Framework for Information Systems Architecture. IBM Systems Journal. vol. 26, 3 (1987)
18. Tepandi, J., Piho, G., Liiv, I.: Domain Engineering for Cyber Defense: a Case Study and Implications. In: CCDCOE Conference on Cyber Conflict, Tallinn, accepted paper (2010)
19. Beck, K.: Test-Driven Development: By Example. Addison-Wesley (2003)
20. Evans, E.: Domain-Driven Design: Talking Complexity in the Heart of Software, Addison-Wesley (2004)
21. Zachman,J. A.: The Framework for Enterprise Architecture – Cell Definitions. ZIFA (2003)
22. Zachman, J. A.: The Zachman Framework: A Primer for Enterprise Engineering and Manufacturing. (2003)
23. ASTM, E1578-06 Standard Guide for Laboratory Information Management Systems (LIMS). ASTM International (2006)

Software Process Improvement

Comparison of Plan-driven and Agile Project Management Approaches: Theoretical Bases for a Case Study in Estonian Software Industry

Marion Lepmets[1] and Margus Nael[2]

[1]Institute of Cybernetics at Tallinn University of Technology
marion.lepmets@ttu.ee
[2]Tallinn University of Technology, Department of Computer Engineering
margus.nael@gmail.com

Abstract. There is evidence that Scrum could benefit from additional plan-driven practices of defined processes that address the organizational context. The question remains however how to add the plan-driven practices without losing the agility of Scrum and increasing the project performance. This paper describes the preparatory phase of the research that aims to evaluate the tailored project management process in industry. First, we compare the plan-driven project management practices derived from the process models of CMMI and ISO/IEC 15504, the PMBoK, project management literature with the Scrum practices from the case company. We then evaluate the comparison addressing the concerns of using Scrum in industry focusing on the value the plan-driven practices would have in agile environment. Finally, we propose a tailored project management process and suggest measures for evaluating the process in an industry case study.

Keywords: Scrum, CMMI, ISO/IEC 15504, PMBoK, project management, plan-driven methods.

1 Introduction

Agile software development is a way of organizing the development process, emphasizing direct and frequent communication – preferably face-to-face, frequent deliveries of working software increments, short iterations, active customer engagement throughout the whole development life-cycle and change responsiveness rather than change avoidance. Agile software development can be seen as a philosophy and several defined methods based on these ideas are in use, all sharing a common set of values and principles. The best known and most used agile methods are Extreme Programming (XP) and Scrum [1]. The goal of agile software development is to increase the ability to react and respond to changing business, customer and technological needs at all organizational levels. Agile software development methods are used in a hope of nearing toward this goal [2]. Agile methods have been applied in industry for many years now without them being given much attention in research [3]. In our paper we are trying to fill a gap by focusing on how agile practices relate to established fields in software engineering. We do that by relating Scrum project management practices of a software development company with the already established practices of project management from process models.

Scrum has attracted significant attention among software practitioners during last five years. Whereas the Extreme Programming method that has been widely accepted as one of the most important agile approaches has a definite programming flavour, Scrum concentrates on managing software projects [4]. Scrum was first described by Ken Schwaber and starts with the premise that software development is too complex and unpredictable to be planned exactly in advance [5]. Scrum is an empirical approach based on flexibility, adaptability and productivity. It leaves open for the developers to choose the specific software development techniques, methods, and practices for the implementation process. It involves frequent management activities aiming at consistently identifying any deficiencies or impediments in the development process as well as in the practices that are used [2]. The environmental and technical variables like time frame, quality, requirements, resources and tools must be controlled constantly in order to be able to adapt to changes flexibly. This is achieved through an iterative and incremental development process.

It is argued that focusing on people will improve software productivity and quality [3]. That is exactly what agile does and why it has become so popular. At the same time, agile methods generally lack practices and guidance for implementing and supporting an agile approach across the organization. It is argued that an agile implementation will not "stick" without an organizational context that supports process definitions that are described in Capability Maturity Model Integration [6]. As Boehm and Turner put it in [7] agility without discipline is the unencumbered enthusiasm of a startup company before it has to turn a profit. The discipline of plan-driven methods approach development with standard, well-defined processes that organizations improve continuously.

Software process improvement (SPI) is an applied academic field rooted in software engineering and information systems disciplines, which has been studied for almost twenty years now. It deals primarily with the professional management of software firms, and the improvement of their practice, displaying the managerial focus rather than dealing with the techniques that are used to write software [8]. SPI is based on process assessment where practices of a software company are evaluated against the requirements and practices of process models. There are currently dozens of process models for assessing and improving software development and its related practices. CMMI (Capability Maturity Model Integration) for Development, ISO 9001, ISO/IEC 15504 (IS 15504), ISO/IEC 12207, ISO/IEC 15288 are only a few of the popular ones. In this study we focused on IS 15504 and CMMI as they are widely used in software industry for process assessment purposes. IS 15504 is the only international standard for process assessment at the moment. CMMI, whose underlying ideas have endured over time, has evolved from the concepts of software maturity framework that was developed by Software Engineering Institute already in 1986 and is used extensively today.

As was stated in [4] Scrum could be tailored to be more compliant with CMMI and CMMI model can be improved by adding some Scrum practices on their activities.

The question remains how to tailor it so that the best practices of the process models are added to Scrum without losing the agility in its software project management.

The aim of this research is to find out whether combining the practices of plan-driven and agile methods in project management will increase project performance. In order to attain the aim of the study, a comparison of project management practices was carried out based on process models of CMMI for Development v.1.2 [9], IS 15504 [10], Project Management Body of Knowledge (PMBoK) of PMI [11] and the project management practices from Scrum environment. The measures of project performance are also suggested which will be used for evaluating the tailored Scrum process in industry. This paper illustrates the first phase of the research in which the bases for tailoring Scrum project management process is provided.

2 Background and Motivation

Scrum is a product-centered and teamwork oriented way of working that does not describe rules or activities about project initiation or finalizing. Most of the tasks in a software development project are assigned and completed within the team. Glazer et al. claim in [6] that agile methods do not have an organizational context that is described in CMMI. Organizational context supports a long-term view of process adaptation across the enterprise as a whole. For example, knowledge sharing on organizational level through project postmortem reviews would increase the project performance of future projects. The goal of the postmortem reviews is to transfer the experiences of project teams into immediate and concrete software process improvements [12]. The retrospective meetings of Scrum are carried out within the team after every iteration, but no retrospective meeting is held after the development project is finished. The postmortem review at the end of the entire development project allows dissemination of project's experiences to wider audience of developers and managers in the organization. As long as the causes of failed projects are not analyzed, understood and communicated, it is unlikely that the project performance can improve in subsequent projects [13].

Plan-driven methods are characterized by heavy upfront planning, focus on predictability and documentation. Scrum, on the other hand, relies on tacit knowledge within a team as opposed to documentation [7]. There are no budget and status reports required in Scrum environment for progress reviews, for example. Although the software department of a company might share the cultural values similar to agile principles, which is necessary for successful adoption of agile methods [14] and the management approve the adoption of agile methods, the members of the board and management might still want to bring themselves up-to-date with the development projects of the company through various progress reports. For mutual understanding about the project progress, there is a need for standardized indication for the effort, time and budget consumed in the project at any certain time. This kind of deliverables like project status reports have been described in detail as a part of project management process in the plan-driven methods.

These examples are valid in our case company and are supported by relevant related literature motivating this study in tailoring Scrum activities with defined processes of project management.

3 Related Research

There is currently a lot of research conducted on software process improvement and plan-driven software development but it is argued in [3] that agile methods have been given little attention in research despite them having gained industrial acceptance. They claim that management oriented approaches such as Scrum is an example of an area where there is a large gap between industrial acceptance and coverage in research. They also suggest research to understand how various practices and recommendations in agile development relate to established fields on various issues like project management.

The current research addresses both the topic of agile software development method Scrum and the established field of project management. The latter is viewed based on literature review, the PMBoK and process models of software process improvement area, described in greater detail in [15]. The detailed practices of Scrum are described based on Scrum rules of Scrum alliance [16] that are followed in software development projects of the case company. We describe below the four research articles that are closely related to this study, focusing on agile or, more specifically, Scrum, and the process models.

Tor and Hanssen describe the potential benefits of combining ISO9001 and agile methods in [1]. They conclude that the main difference between ISO 9001 and agile methods is that ISO 9001 insists on documentation for reviews and to demonstrate process conformity. Agile methods try to avoid writing documents that does not contribute to the finished system. On the other hand – if the customer requires a certain document, the use of agile methods are no hindrance for developing them.

Glazer et al. have written a report [6] about embracing both CMMI and agile where they claim that combining the benefits of agile and CMMI will dramatically improve business performance. The report describes in detail CMMI and the agile methods, providing a thorough paradigm comparison and concludes that practitioners should make the best use they can of both of these paradigms to encounter the betterment of their project and organization. This report is a general description of the paradigms and is not aiming to describe any agile method in particular. After having described both paradigms, it suggests that there should be benefits in using a combination of them.

Marçal et al. have mapped the project management practices of CMMI and Scrum in [4]. Their paper shows how Scrum addresses the process areas of project management of CMMI.

Turner and Jain described the results of a workshop activity in [17] where the characteristic of two approaches, agile and CMMI were compared, resulting in a broad mapping between the two approaches. The comparison was made by a group whose members had expertise in both agile and CMMI and remains on the same level of abstraction as the paper by Marçal et al.

These mappings provide valuable insight into the differences and similarities of the two paradigms. Our study differs from the previous ones as we take the comparison from the conceptual to the practice level.

4 Research Method

This study is characterized as analytical and evaluative research and it mainly follows the constructive research approach. Based on [18] there are two processes in the constructive approach - first, the process of building a construct, artifact or model; and second, the process of determining how well the construct, artifact or model performs. In this study we compare the project management practices, tailor the Scrum process and set the measures for evaluating the tailored process in industry case studies.

In this research the motivation comes from industry where additional plan-driven practices need to be combined with their Scrum practices in order to improve their estimation accuracy and project performance in software development.

The case company in our study is a medium-sized [19] software development company that has applied Scrum systematically from 2008 and in four development projects since. The average size for the team has been five people and the average size of project has been six months. A survey among the software developers and project managers in the company has resulted in a set of concerns while using Scrum and has motivated the study of comparing the Scrum practices with the plan-driven practices. The questions were open-ended and targeted to the impediments or shortages of Scrum. The survey results indicate that the information was kept in Scrum teams instead of sharing the knowledge across different organizational levels. This led to the developers missing important technical knowledge and the managers not knowing all the project details. The survey findings suggest that the biggest impediment of the team-centered Scrum process is that it does not support information spreading outside the project.

5 Comparison of Scrum and Plan-driven Project Management Practices

In this chapter we describe the basis of comparison between the project management practices. First, the Scrum practices have been described according to Scrum rules of Agile alliance that are implemented in the software development projects of our case company. We then illustrate the set of plan-driven project management practices that have been derived from CMMI, IS 15504, the PMBoK and project management literature and grouped together in [15].

5.1 Scrum Practices

In this chapter we describe Scrum through its rules, roles and artifacts. We also describe the project management practices of Scrum that are based on the case company using Scrum in their development projects.

Scrum is a product-centered people-oriented framework for project management. It applies iterative and incremental approach to optimize the risks and predictability. The Scrum process is transparent and its frequent deliveries and inspections ensure the highest predictability. The Scrum framework includes roles, rules, artifacts and time-boxes. Time-box is a period of time in which to accomplish a task. Every role and activity in Scrum is time-boxed and thus it improves the velocity of the work. Every role and activity has a set of rules that together will support the achievement of the expected results of the project. Thus, the rules connect the roles, artifacts and time-boxes.

There are three roles in Scrum: the ScrumMaster, the Product Owner and the team. ScrumMaster is responsible for managing the Scrum process. He coaches and leads the team to become better in Scrum, but nonetheless the team is self-organizing and ScrumMaster is not the head of the team like a project manager in plan-driven development project.

Product Owner is the only person responsible for managing the product feature list. He drives the product vision and is the only person who tells the team what to do. A typical Product Owner is either a service, product or business manager.

The team is cross-functional set of developers who turn the product feature list into potentially shippable set of functionalities. The team is self-organizing, i.e. it finds the best way all by itself how to turn the list of features into a shippable product while facing the impediments and challenges as one. The best team size of Scrum is from five to nine persons.

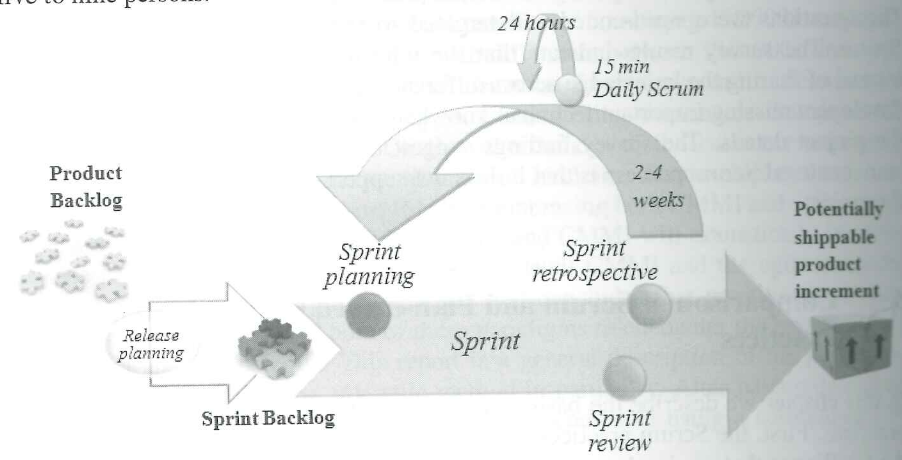

Fig. 1. The Scrum process (*activities in Italic, artifacts in Bold*)

The Scrum process illustrated on Fig.1 consists of five time-boxed meetings and the work period itself. The process is iterative and the iterations are called sprints. The product is developed iteratively, wherein each sprint delivers a set of highest priority and riskiest features or functionality of the product. The product is released when enough sprints have been done, i.e. when enough value has been added sprint by sprint to the product.

In the beginning of the project or release there is a release planning meeting where the team, the Product Owner and ScrumMaster participate. The purpose of the meeting is to set up a release plan and it does not take more time than 15-20% of the time that an organization typically consumes to build a release plan.

The iterative part of the project is the sprint, which contains the following activities: the sprint planning meeting, the development work, the sprint review and the sprint retrospective. The sprint planning meeting is held to set the sprint goal, i.e. what and how the development work will be carried out. The sprint goal is the subset of the release goal and is set according to the data of various inputs like the product backlog, the latest increment of the product, the capability and the past performance of the team.

The product backlog is a list of all features, technologies, functions and improvements that will be made to the product. Product backlog is evolving constantly through the time. Every item in the product backlog is described through three attributes: the description of the item, priority of the item and an estimate for the item being developed.

Only the team decides what and how the work is done in the sprint. Sprint review is an informal meeting where the team and stakeholders describe what was done and what should be done next. The sprint retrospective meeting is for process improvements as the team decides what should be done better in the next sprint. During the development process there are daily stand-up meetings of 15 minutes called the daily Scrums. The purpose of the daily Scrums is to improve communication and transparency. Every team member tells what he accomplished since the last meeting, what he will do before the next meeting and what the impediments on the way are.

In addition to roles and time-boxed activities, there are four artifacts: the product backlog (PB), release burndown chart (RBD), the spring backlog (SB) and the sprint burndown chart (SBD). Product backlog lists the features of the product. Release burndown chart illustrates graphically the sum of remaining effort according to the estimates in the product backlog. The Product Owner is responsible for updating the graph throughout the project. Sprint backlog lists all the tasks that the team performs to reach the sprint goal and the sprint burndown chart describes the effort remaining for the goal to be reached.

According to the company in our case study, the Scrum practices can be listed based on the tasks of different Scrum roles and meetings. The numbers have been added for each listed practice to make the grouping of practices easier for the comparison.

1. ScrumMaster (activities of)
1.1. Monitor sprint work (through updating sprint burndown chart)
1.2. Schedule control (through updating sprint burndown chart)
1.3. Remove organizational impediments that impede the Scrum
2. Product Owner
2.1. Create and maintain the product backlog (list of features and tasks during product development)
2.2. Set priorities to every item in the product backlog
2.3. Set the acceptance criteria to every item of the product backlog
2.4. Do the acceptance testing to the shipped product
2.5. Update the product release burndown chart
3. Release planning
3.1.Prioritize the product backlog

3.2. Estimate the product backlog
3.3. Set the overall features and functionality that the release will contain
3.4. Set the goals and establish a plan for the release
3.5. Define major risks
3.6. Define probable delivery date and cost
3.7. Establish Scrum rules
4. Sprint planning
4.1. Set the sprint goal, i.e. what will be done in the sprint
4.2. Establish the spring backlog (list of task in the sprint)
4.3. Identify tasks for the sprint
4.4. Design the sprint work
4.5. Define activities to achieve the sprint goal
5. The sprint
5.1. Develop the product
5.2. Test the product
5.3. Documentation
5.4. Review of estimated remaining work (by Team members)
6. Sprint review
6.1. Product Owner acceptance tests the increment of the product
6.2. Team suggests what to do next
6.3. Review the BurnDown chart (by Product Owner)
7. Sprint retrospective
7.1.Create a prioritized list of the major items of success in the sprint and how to improve in the next sprint
7.2. Create a prioritized list of the major items of impediments for the team and how to remove them
8. Daily Scrum meetings
8.1. Find current risks and impediments

5.2 Plan-driven Project Management Practices

The basic project management practices of process models have been combined and described in [15] that form a part of the theoretical bases for the current study. The project management and related practices were combined there from CMMI and IS 15504. Project management activities were added from the PMBoK and from over 12 sources of project management literature. The set of basic project management activities described in [15] is viewed as the set of plan-driven project management practices in the current study.

The project management practices from process models can be viewed as best practices. They are the specific practices from the Project Planning and Project Monitoring and Control process areas of CMMI, and the Project Management base practices from IS 15504. Project management activities of the PMBoK and project management literature that are not described in the process models are also added to the set of plan-driven project management practices. The final set of plan-driven project management practices is grouped together with Scrum practice in chapter 5.3 in Table 1. The method followed in grouping these practices has been described in greater detail in [15].

5.3 Grouping the Plan-driven and Scrum Project Management Practices

The following table (Table 1) shows the grouping of the plan-driven and Scrum project management practices. A row without a corresponding Scrum practice indicates that the practice is not described in Scrum on explicit practice level and is therefore unknown to Scrum.

Table 1. Grouping the plan-driven and Scrum project management practices

IS 15504 *Project Management*	**CMMI** *Project Planning, Monitoring & Control*	**Scrum**
MAN.3.BP1: Define the scope of work	SP 1.1 Estimate the scope of the project	2.1 Create and maintain PB
MAN.3.BP2: Define project life cycle	SP 1.3 Define project life cycle	3.7 Establish Scrum rules
MAN.3.BP3 Evaluate feasibility of the project		
MAN.3.BP4: Determine and maintain estimates for project attributes	SP 1.2 Establish estimates of work products and task attributes SP 1.4 Determine estimates of effort and cost	3.1 Prioritize PB 3.2 Estimate PB 4.1 Set the sprint goal 2.3 Set the acceptance criteria to items of PB
MAN.3.BP7: Define project schedule	SP 2.1 Establish project budget and schedule	3.2 Estimate PB 3.6 Define probable delivery date and cost
	SP 2.2 Identify project risks	3.5 Define major risks
MAN.3.BP5: Define project activities and tasks		4. Sprint planning
MAN.3.BP6: Define needs for experience, knowledge and skills	SP 2.5 Plan for needed knowledge and skills	
	SP 2.4 Plan for project resources	
	SP 2.3 Plan for data management	
MAN.3.BP8: Identify and monitor project interfaces		
MAN.3.BP9: Allocate responsibilities	SP 2.6 Plan stakeholder involvement	
MAN.3.BP10: Establish project plan	SP 2.7 Establish the project plan	4.1 Set the sprint goal 4.2 Establish SP 4.3 Identify tasks for the sprint
MAN.3.BP11: Implement the project plan		4.4 Design the sprint work 4.5 Define activities to achieve the sprint goal 5. The sprint
	SP 3.1 Review plans that affect the project	
	SP 3.2 Reconcile work and resource levels	

	SP 3.3 Obtain plan commitment	4.2 Establish SB
MAN.3.BP12: Monitor project attributes	SP 1.1 Monitor project planning parameters	1.1 Monitor sprint work 1.2 Schedule control 2.5 Update RBD
	SP 1.2 Monitor commitments	1.1 Monitor sprint work
	SP 1.3 Monitor project risks	8.1 Find current risks and impediments 7.2 List impediments
	SP 1.4 Monitor data management	
	SP 1.5 Monitor stakeholder involvement	
MAN.3.BP13: Review progress of the project	SP 1.6 Conduct progress reviews	6.3 Review SBD 5.4 Review of estimated remaining work
	SP 1.7 Conduct milestone reviews	6. Sprint review
	SP 2.1 Analyze issues	7.2 List impediments
		7.1 List success factors
MAN.3.BP14: Act to correct deviations	SP 2.2 Take corrective action	5. The sprint
	SP 2.3 Manage corrective action	1.3 Remove organizational impediments that impede the Scrum
Project finalizing activities from the PMBoK:		*Scrum*
Information to formalize project completion is gathered and disseminated		
The project is evaluated after closing		
The lessons learned are compiled for future projects		
Project Directing activities from the project management literature:		*Scrum*
Directions are given to the project team		Covered by ScrumMaster and daily Scrum meetings
Supervise the project team		
Motivate the people in the project team		
Coordinate the interactions between the people in the project team		
Explain the reasons behind the decision-making to the project team		
Resolve the conflicts within the project team		

According to the grouping in the table above (Table 1), the following practices were unknown to Scrum: evaluating the feasibility of the project, planning and monitoring the data management, planning for needed knowledge and skills, resource planning and stakeholder involvement planning, stakeholder involvement monitoring, reviewing plans that affect the project, reconciling work and resource levels, the project finalizing activities of the PMBoK and project directing activities from the project management literature.

Most of the plan-driven practices that are unknown to Scrum are covered either by the responsibilities of the ScrumMaster (evaluating the feasibility of the project, planning for needed knowledge and skills, resource and stakeholder involvement planning and monitoring, project directing) or in the daily Scrum meetings (reconciling work and resource levels, reviewing plans that affect the project) but they have not been explicitly described as practices.

The unknown practices of Scrum not covered by ScrumMaster responsibilities nor addressed in daily Scrum meetings are about planning and monitoring the data management described in CMMI and project finalizing described in the PMBoK. Data management includes the processes and systems that plan for, acquire and provide stewardship for business and technical data throughout the data life-cycle [9]. The practice of data management addresses the organizational level and requires rigorous planning and monitoring that is too heavy for an agile method. The practices and rules defined in Scrum contribute for good communication and promote collaboration between team and stakeholders, project information is shared in meetings or document available to everyone [4]. Scrum relies on the tacit knowledge rather than documentation [7].

At the same time, Scrum would benefit from the postmortem analysis carried out in the end of the project described in the PMBoK. The causes of failed projects should be analyzed, understood and communicated in order to improve the performance of subsequent project [13].

The only unknown practice for the plan-driven methods from Scrum is one of the practices from sprint retrospective - prioritize major items of success in the last sprint and describe how to improve in the next sprint. According to CMMI, project related issues and impediments are analyzed, corrective action is taken and monitored. Scrum suggests carrying not only the impeding but also the success factors into the next sprint to increase a chance to improve immediately. Similar practice could be added to the plan-driven practices.

5.4 Tailoring the Scrum Process

The comparison of practices described in Table 1 illustrates that Scrum is not targeting the organizational level practices. As was stated by Turner and Jain in [17] the scope of agile approach and of CMMI differs. CMMI has broad, inclusive and organizational approach while agile has small and focused approach. Sutherland et al. also found in [19] that CMMI has a concept of institutionalization that can help establish needed discipline to adopt agile methods organization wide. According to [6] the focus of the agile paradigm is on project and team while CMMI is implemented at organizational level. Therefore Scrum project management could be improved by adding practices that address organizational level or enhance the communication between project and organizational level.

The CMMI specific practices about planning the project resources and for the needed knowledge and skills are handled in the case-company during the project preparation phase but they are not addressed in Scrum rules and activities or by any Scrum roles. In addition to that, Scrum could benefit from postmortem reviews corresponding to the three unknown practices of Scrum from the PMBoK. Also, the output work products of project progress review described in process models could contribute to better managerial overview of Scrum project status during project development. All these improvements are pointing towards organizational level shortage.

CMMI [9] describes the typical work products of progress and milestone reviews as documented progress or milestone review results. IS 15504 describes the progress status record in detailed level saying that it has the following possible attributes: the status of actual tasks against planned tasks, status of actual results against established goals, status of actual resource allocation against planned resources, status of actual cost against budget estimates, status of actual time against planned schedule, status of actual quality against planned quality and record of any deviations from planned activities and reasons why (15504-5).

In order to enhance the communication between project and organization levels, the progress status reports could be added to the tailored Scrum process as described in Figure 2.

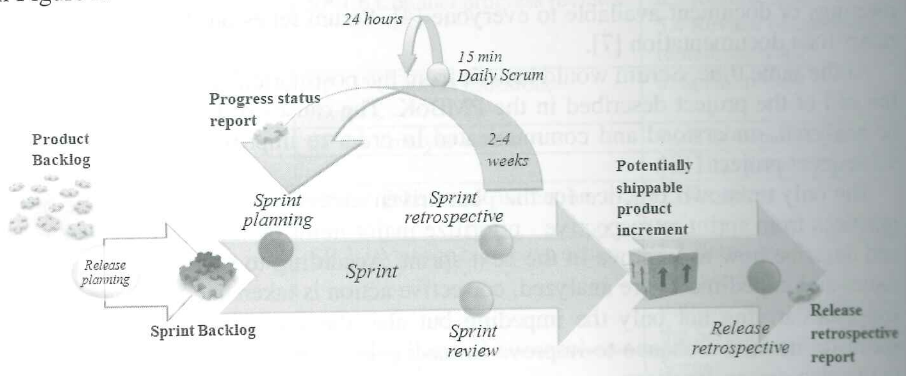

Fig. 2. The tailored Scrum process (*activities in Italic, artifacts in Bold*)

The progress status reports are periodical reports written by either ScrumMaster or Product Owner and distributed to the team and management. The progress status report includes the description of the status of tasks, resources, costs, schedule and quality against their estimates.

In addition to the progress status report that has been added to the tailored Scrum process, the project finalizing activities of the PMBoK unknown to Scrum are also added to support the increase in project performance in subsequent projects. The three PMBoK practices are: information to formalize project completion is gathered and disseminated, the project is evaluated after closing and the lessons learned are compiled for future projects. The new practice for the tailored process is called *release retrospective* and will include similar tasks of sprint retrospective, i.e. create a prioritized list of major items of success in the release and how to improve in subsequent releases; and create a prioritized list of major items of impediments in the release and how to remove them from subsequent release. These tasks correspond to the PMBoK practices of project evaluation and compiling the lessons learned. The release retrospective has an output artifact of *release retrospective report* which is shared among the team, posted in the collaboration tool of the company and emailed to all ScrumMasters and Product Owners for organizational knowledge sharing purpose. The ScrumMaster and the Product Owner of the release share the responsibility for the practice to be carried out and artifact delivered. The sharing of

the release retrospective report corresponds to the PMBoK practice of being gathering and disseminating information of project completion.

6 Conclusions and Future Work

In our research, we compared the plan-driven project management practices with Scrum project management practices and tailored the industry project management process so that it would address the organizational level.

Next, we plan to measure the increase of estimation accuracy and project performance of the tailored process. In order to do that, the tailored process will be implemented in industry for evaluation purposes. The data will be collected about project performance and estimation accuracy. According to Salo and Abrahamsson [12], the main problem with agile development is the lack of detailed planning of the iteration, namely the effort estimation. Project estimation accuracy should increase when the underlying causes for the gaps between the estimated and realized effort for each task have been realized. With the progress status reports and release retrospective reports the estimation accuracy should increase. The estimation accuracy will be followed after implementation of the tailored process and compared to the earlier estimation results.

Project performance will be measured through both quantitative and qualitative measures collected in industry cases. The project performance factors used in this study are described in [21] - the project's ability to meet budget commitments, ability to meet schedule commitments, ability to achieve customer satisfaction, ability to meet the defined goals, productivity in the project, and the product's ability to satisfy specified requirements. The data on budget, schedule, project and product goals are collected in the case company and will be compared to the data prior to implementing the tailored process. Customer satisfaction surveys are carried out in each project and are compared to see whether the tailored process has increased the customer satisfaction. The project productivity is measured through project velocity that is described in [12] where the increased project velocity corresponds to increased project productivity.

Acknowledgments

This research was supported by the European Regional Development Fund through the Estonian Center of Excellence in Computer Science, EXCS.

References

1. Stålhane T., Hanssen G.K.: The Application of ISO 9001 to Agile Software Development. In: Product-Focused Software Process Improvement 9th International Conference, PROFES 2008. LNCS, vol. 5089, pp. 371-385. Springer, Heidelberg (2009).

2. Abrahamsson P., Warsta J., Siponen M. T., Ronkainen J.: New Directions on Agile Methods: A Comparative Analysis. In: 25th IEEE International Conference on Software Engineering, 244p. Portland, Oregon (2003).
3. Dingsøyr T., Dybå T., Abrahamsson P.: A Preliminary Roadmap for Empirical Research on Agile Software Development. Agile 2008 Conference, 83-94 (2008)
4. Marçal, A. S. C.; Freitas, B. C. C.; Soares, F. S. F.; Belchior, A. D.: Mapping CMMI Project Management Process Areas to SCRUM Practices. (2008). www.cesar.org.br/files/file/SCRUMxCMMMI_IEEE-final03.pdf Accessed 09.10.2009.
5. Controlled Chaos, www.controlchaos.com/old-site/ap.htm, Accessed 09.10.2009.
6. Glazer H., Dalton J., Anderson D., Konrad M., Shrum S.: CMMI or Agile: Why Not Embrace Both! CMU/SEI-2008-TN-003, Software Engineering Institute (2008), 41p.
7. Boehm B. and Turner R.: Balancing Agility and Discipline - A Guide for the Perplexed. Pearson Education, Boston (2004), 266p.
8. Hansen B., Rose J., Tjornehoj G.: Prescription, description, reflection: the shape of the software process improvement field. International Journal of Information Management, Vol. 24. 457-472 (2004)
9. CMMI for Development. CMU/SEI-2006-TR-008, ESC-TR-2006-008, Version 1.2, CMMI Product Team, Software Engineering Institute (2006), 537p.
10. ISO/IEC 15504-5. ISO/IEC 15504-5 Information Technology - Process Assessment - Part 5: An Exemplar Process Assessment Model, 1st edition, ISO/IEC JTC1/SC7 (2006), 162p.
11. Project Management Body of Knowledge: A guide to Project Management Body of Knowledge. Project Management Institute, Pennsylvania, 209p (2000)
12. Salo O. and Abrahamsson P.: An Iterative Improvement Process for Agile Software Development. Software Process Improvement and Practice, Vol. 12, 81-100 (2006)
13. Verner J. M. and Evanco W. M.: In-house Software Development: What Software Project Management Practices Lead to Success. IEEE Software (January/February), 86-93 (2002)
14. Tolfo C., Wazlawick R. S., Ferreira M. G. G., Forcellini F. A.: Agile Methods and Organizational Culture: Reflections about Cultural Levels. Software Process Improvement and Practice. (2009). http://www3.interscience.wiley.com/cgi-bin/fulltext/122613722/PDFSTART, Accessed 20.10.1009.
15. Lepmets M.: Evaluation of Basic Project Management Activities - Study in Software Industry. Tampere University of Technology, Publication 699, Pori, Finland (2007), 223p. http://dspace.cc.tut.fi/dpub/bitstream/handle/123456789/68/lepmets.pdf?sequence=1
16. Scrum Alliance (2009), http://www.scrumalliance.org/, Accessed 12.10.2009.
17. Turner R. and Jain A.: Agile Meets CMMI: Culture Clash or Common Cause? XP/Agile Universe 2002. In: Wells D. and Williams L. (eds.). LNCS, vol. 2418, pp. 153-165. (2002)
18. Järvinen P.: On Research Methods. Juvenes Print, Tampere, Finland (2001), 190p.
19. EC SME definition, http://ec.europa.eu/enterprise/enterprise_policy/sme_definition/index_en.htm, Accessed on 12.10.2009.
20. Sutherland J., Jakobsen C. R., Johnson K: Scrum and CMMI Level 5: The Magic Potion for Code Warriors. Agile Conference. Denver, CO (July 2005) http://jeffsutherland.com/scrum/Sutherland-ScrumCMMI6pages.pdf, Accessed 05.09.2009.
21. Goldenson, D. R. and Herbsleb J.: After the Appraisal: A Systematic Survey of Process Improvements, Its Benefits, and Factors that Influence Success. Technical Report CMU/SEI-95-TR-009, Software Engineering Institute (1995). 66p.

An Implementation of Self-Testing

Edgars Diebelis, Prof. Dr. Janis Bicevskis

Datorikas Institūts DIVI, A.Kalniņa str. 2-7, Rīga, Latvia
Edgars.Diebelis@di.lv, Janis.Bicevskis@di.lv

Abstract. This paper is devoted to the analysis of advantages and implementation mechanisms of self-testing, which is one of the smart technologies. Self-testing contains two components: full set of test cases and built-in testing mechanism (self-testing mode). The test cases have been collected since project start and they have been used in integration, acceptance and regression testing. The built-in self-testing mode provides execution of test cases and comparison of test results with saved standard values in different environments. This paper continues the approach described in article Self-Testing - New Approach to Software Quality Assurance, expanding it with the concept of a test point, which allows flexible defining of execution points of testing actions. Furthermore, the paper describes the first implementation of self-testing using test points.

Keywords: Testing, Smart technologies, Self-testing.

1 Introduction

The self-testing is one of the features of smart technologies [1]. The concept of smart technologies proposes to equip software with several built-in self-regulating mechanisms, which provide the designed software with self-management features and ability to react adequately to the changes in external environment similarly to living beings. The necessity of this feature is driven by the growing complexity of information systems and the fact that users without profound IT knowledge can hardly use such complex systems. The concept of smart technologies besides a number of significant features also includes external environment testing [2, 3], intelligent version updating [4], integration of the business model in the software [5]. The concept of smart technologies is aiming at similar goals as the concept of autonomous systems developed by IBM in 2001 [6, 7, 8]. Both concepts aim at raising software intellect by adding a set of non-functional advantages - ability to adapt to external situation, self-renewing, self-optimizing and other advantages. However, features and implementation mechanisms of both concepts differ significantly. The autonomous systems are built as universal and independent from properties of a specific system. As a rule, they function outside of a specific system and cooperate on the level of application interface. Hence, we may consider the autonomous systems being more like environmental properties than specific systems. Whereas the features of smart technologies provide a scaffolding, which is filled with functional possibilities of a specific system, thus integrating the implementation modules of smart technologies with the modules of a specific system. Therefore, further development of both concepts is highly valuable.

The first results of practical implementation of smart technologies are available. Intelligent version updating software was developed and is used in practice in a number of Latvian national-scale information systems, the largest of which, FIBU, manages budget planning and performance control in more than 400 government and local government organisations with more than 2000 users [4]. Firstly, external environment testing [3] is used in FIBU, where the key problem is the management of operating systems and software versions for the large number of territorially distributed users. Secondly, external environment testing is employed by the Bank of Latvia in managing operations of many systems developed independently.. The use of smart technologies has proved to be effective in both cases [9]. The third instance of the use of smart technologies is the integration of a business model and an application [5]. The implementation is based on the concept of Model Driven Architecture (MDA) [10], and it is used in developing and maintaining several event-oriented systems. The use of smart technologies has been proven to be effective according to the results obtained in practical use. This study continues the research of the applicability of smart technologies in software testing.

Self-testing provides the software with a feature to test itself automatically prior to operation; it is similar to how the computer itself tests its readiness for operation when it is turned on. By turning on the computer self-testing is activated: automated tests are run to check that the required components, like hard disc, RAM, processor, video card, sound card etc, are in proper working order. If any of the components is damaged or unavailable, thus causing operation failure, the user receives notification. The purpose of self-testing is analogical to turning on the computer: prior to using the system, it is tested automatically that the system does not contain errors that hinder the use of the system.

The paper is composed as follows: To explain the essence of the self-testing approach, the first section repeats in brief the ideas on the self-testing method and deals with its modes [11, 12]. Section 2 looks at the approach for determining the state of database prior to system self-test, and Section 3 deals with the concept of test point, and Section 4 describes in brief the technical implementation of self-testing.

As of writing this paper, the first version of the self-testing software has been developed; it contains a test control block (test execution and test result control) and a self-testing software library, which contains the functions included in the system to be tested. Technology of self-testing could be approbating for different applications. Now the self-testing technology is being approbated for using in securities and currency accounting system applications in banking.

2 Method of Self-Testing

The main principles of self-testing are:

- Software is delivered together with the test cases used in automated self-testing;
- Regression testing of full critical functionality before every release of a version;
- Testing can be repeated in production, without impact on the production database.

As shown in [11, 12], self-testing contains two components:

- Test cases of system's critical functionality to check functions, which are substantial in using the system;
- Built-in mechanism (software component) for automated software testing (regression testing) that provides automated executing of test cases and comparing the test results with the standard values.

The defining of critical functionality and preparing tests, as a rule, is a part of requirement analysis and testing process. The implementation of self-testing requires at least partial inclusion of testing tools functionality in the designed system. The implementation of self-testing functionality results in complementing the designed system with self-testing functionality calls and a library of self-testing functions (.dll file). Certainly, the implementation of self-testing features requires additional efforts during the development of the system. However, these efforts are justified by many advantages obtained in development and in long-term maintenance of a high quality system in particular.

The main feature of self-testing is ability to test the software at any time in any environment - development, test and production environments. While developing the mechanism of self-testing the developers may not enter the information into production database; however, they can be used in read-only mode. Hence, it is possible to implement testing in test or production environment without any impact on system use. Of course, it is useful to complement the set of tests with recent system modifications to ensure that testable critical functionality in self-testing is covered.

2.1 Self-testing software

The self-testing software is partly integrated in the testable system, which has several operating modes; one of them is self-testing mode when an automated execution of testing (process of testing) is available to the user. After testing, the user gets a testing report that includes the total number of tests executed, tests executed successfully, tests failed and a detailed failure description. The options provided by self-testing software are similar to the functionality of testing support tools.

2.2 Phases of system testing

In order to ensure development of high quality software, it is recommendable to perform testing in three phases in different environments [11]:

- Development environment - in this environment the system has been developed, errors are corrected and system patches are made;
- Test environment - this environment is used to test error corrections and improvements. In order to replicate situations in the production environment in test environment, at least, for example, once a month production environment should be renewed from a backup in test environment;

- Production environment - this environment is used by the system users. Patches and improvements are set only after obtaining successfully testing results in development and test environments.

Testing phases are described in detail in the article Self-Testing - New Approach to Software Quality Assurance [11].

2.3 Modes of self-testing

As shown in [11], the self-testing functionality can be used in the following modes:

- Test storage mode. In this mode, new test cases are defined or existing test cases are edited/deleted. The system logs all necessary information of reading-writing and managing actions by recording them into the test storage file. To provide the self-testing mode in the production environment, an additional database, in which test cases are registered and executed, is used. In the case of production environment, during test storage mode the real database is accessible in the read-only mode. Neither development, nor testing environment requires an additional database, since one database is used for both storing and playing back tests. The results of system operation are stored in the test storage file. Moreover, users can use this mode to report bugs – the user can record the failed test case and forward it together with the description of error to the developer. As a rule, test cases are made according to the developer's interpretation of software specification. In the course of time, the amount and content of test cases increases due to system's evolution.

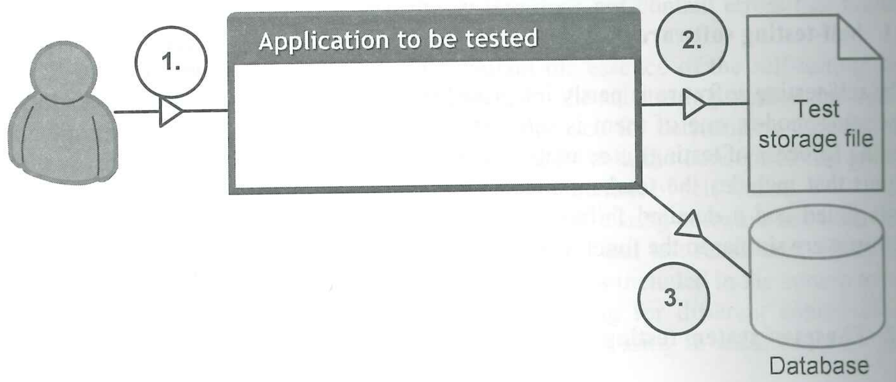

Fig. 1. Test Storage Mode

1. The user registers a test case in the test storage mode. The user uses the same application that is used for daily business purposes.
2. In the test storage mode, the application registers in the test file the actions performed in the system.
3. If data storing is performed in the application, the data are stored in the database. In the case of production environment, the related data are read from the real data-

base, while the storing is done in the test registration and execution database, not in the real database.

- Self-testing mode. In this mode, automated self-testing of the software is done by automatically executing the stored test cases. Test input data are read from the test file. In the development and testing environments, test cases are executed in one database. In the production environment, the test storage and execution database is used. In the self-testing mode, the real database of the production environment is accessible in the read-only mode

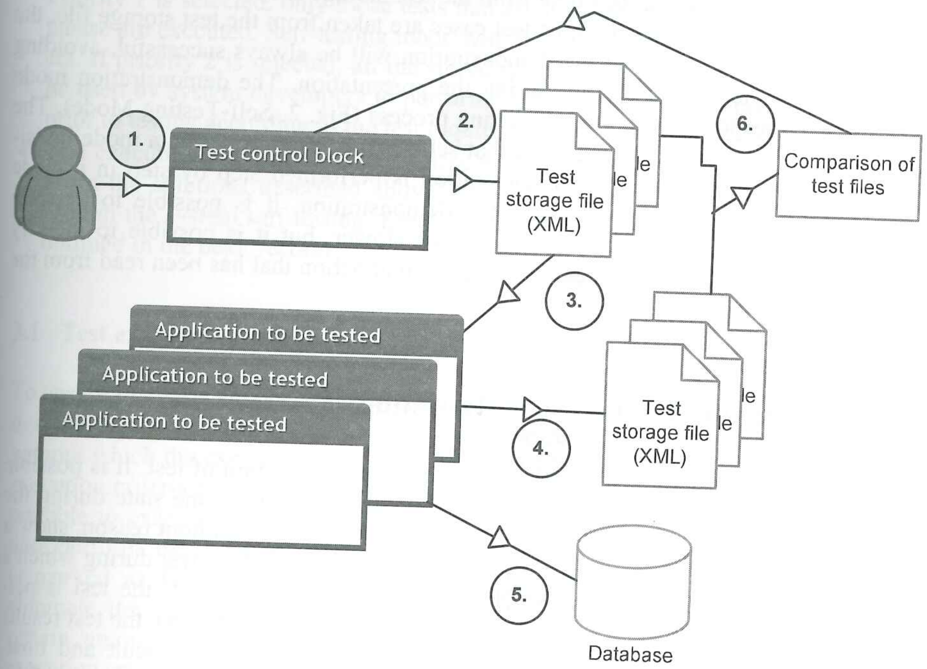

Fig. 2. Self-Testing Mode

1. To call system self-test, the user opens the test control block window.
2. The user loads the list of the available test files and selects the tests files to be executed.
3. The test control block reads information from the test file and executes it; the actions specified in the test file are performed.
4. In the self-testing mode, similarly to the test storage mode, a test file is created. The test file is created using the same approach as in the test storage mode.
5. If data storing is performed in the test case, the data are stored in the database. In the case of production environment, the related data are taken from the real database, while the storing is done in the test registration and execution database, not in the real database.
6. After the testing, the file created in the test storage mode is compared with the file created in the self-testing mode. If the contents of the files match, the test case has

been successful; if they do not match, the test case has failed. Information about test results is displayed in the user's test control block, where, if the test case has failed, the user can find detailed information about the reasons of the failure.

- Use mode. In this mode, there are no testing activities – the user simply uses the main functionality of the system.
- Demonstration mode. The demonstration mode can be used to demonstrate system's functionality. User can perform system demonstrations, by using stored test cases in test storage files. During demonstration mode, form fields are automatically filled with test data from the test storage file, thus demonstrating the functionality of the system. Since test cases are taken from the test storage file, the demonstrator may rely that the demonstration will be always successful, avoiding any inconveniences and errors during the presentation. The demonstration mode process corresponds with the self-testing process (Fig. 2. Self-Testing Mode). The difference between the processes is that self-testing is performed in a mode invisible to the user, whereas the demonstration is performed step by step in a mode visible to the user. Together with the demonstration, it is possible to perform system self-testing. This approach is much slower, but it is possible to identify errors in a visible mode, executing the particular action that has been read from the test file.

3 Preparing the Database for Re-testing

The state of database during the test is crucial for the execution of test. It is possible that the state of database during the executions will differ from the state during the test storing. It means that there are cases where the test might, without reason, show a failure due to changes in the state of the database. For example: a test during which a certain amount is debited from the client's account is registered. If the test is performed repeatedly, it is possible that due to changes in account balance the test results will show a failure. Due to constantly changing database it is difficult and time-consuming to ensure that the stored test is executed with the same state of the database as of test registration moment. Solutions for executing re-tests, considering the variability of database, are described below:

- Creating a backup database. Prior to storing every test, a backup database is prepared, and a backup database is installed prior to executing every test. The backup is stored for as long as the stored tests, which have been registered using the particular backup, are being employed. This approach requires a lot of time and work resources;
- Generation of reverse tests. For every registered test case the self-testing software automatically generates a reverse test that reverses the database to the initial state. For example: If the user registers a test during which a certain amount is debited from the client's account, the self-testing software automatically generates a test that credits the particular amount to the client's account;
- Registration of reverse tests. The aim of the solution is to manage the registration of user tests so that they would contain a full set of events. The self-testing software controls actions of the user who registers the test case, making the user to provide, with initial test cases, a data set with which the user later registers other test cases;
- Consecutive execution of all tests. When the self-testing functionality is implemented in the system, the database backup is taken. There after, a new backup is not taken from the database. Every time when system self-testing is performed, the taken database backup is installed and all the registered tests are performed consecutively;
- Priorities of self-testing mode. Priorities can be selected in the self-testing mode. If Priority 1 is selected, only those tests that are not dependant of the state of the database are executed. Self-testing mode with Priority 1 would be used by system users. If priority 2 is selected, all the stored tests will be executed. This priority will be used by system developers to perform testing, and they will prepare a database prior to testing according to the test requirements;
- Test execution criteria. Criteria of successful test execution are built in the system. From the solutions described above, the test execution criteria approach will be used in the system self-testing. Key considerations for the use of this approach are outlined in the next Section.

3.1 Test execution criteria

To ensure that tests are not dependant of the data set on which the tests are performed, it is necessary to control the key criteria for successful execution of the test and without which the execution of the test is impossible. The control is ensured with test execution criteria built in the system under test, which during the test check that it is possible to execute the specified criterion. If the criterion specified in the test point does perform, the test continues to execute; if not, the test execution is terminated and is marked as failed. The reason of termination is notified to the user, who can eliminate the failure and execute the test again. For example: a test case where a certain amount is debited from the client's account. In this test, the following execution criteria could be implemented and controlled:

- The amount specified in the test case is available in the client's bank account;
- The bank account specified in the test case is registered in the name of the client;
- The client's bank account is not closed.

Advantages of this approach:

- When the test is being executed, it is not necessary to provide the same state of the database it was in when registering the test;
- The solution is not time-consuming. To execute tests, a database backup need not be installed;
- It is possible to execute a particular test(s) not executing all the registered tests;
- The technical implementation is comparatively simple.

The major drawback of this solution is the extra work to be done to implement the test execution criteria in the software.

Hereinafter the paper deals with the Test Point concept. Test execution criteria in the self-testing software are realised as test points, which are in line with the general approach for realisation of self-testing.

4 Test Point

A test point is a command upon which system testing actions are executed. To be more precise, a test point is a programming language command in the software text, prior to execution of which testing action commands are inserted. A test point ensures that particular actions and field values are saved when storing tests and that the software execution outcome is registered when tests are executed repeatedly. By using test points, it is possible to repeat the execution of system events.

As described in the sections above, the self-testing features are introduced in the tested system, namely - written by the test points, which can be introduced in the system in at least two ways:

- By altering the system software's source code. When developing the system, the developer implements in the software code also test points that register system's actions.
- The specialist who defines the business process schemes specifies the test points in the business process. In this case, the business processes and the software must be compatible, and extra resources for moving the testing actions to the software are required. For the time being, the authors do not have knowledge of any instances of application of the approach described above in practice.

When initially developing the self-testing software concept, it was planned to develop only test points that ensure the registration of data storage in the database and data selection from database events. It was important to check whether when executing repeatedly a database command (INSERT, UPDATA, SELECT, procedure or function call etc), the result saved in the database or selected from the database matches the data storing or data selecting performed in the first time.

While evolving the self-testing concept, the idea to use the test point approach to register all system events emerged. Thus, test points register not only data storing in database events or data selection from database events but also other application events (filling in fields in application form, calling application events etc). Such changes ensure that user interface and business logics are tested as well; also, this approach provided a possibility for users to use the system in the demonstration mode. Consequently, with comparatively low investments, the functionality of self-testing was increased considerably.

To show how test points are used, a stock purchase transaction process is shown in the figure below (Fig. 3. Stock Purchase Transaction Process). The registration of a stock purchase transaction consists of the following main steps:

- Specifying the client;
- Selecting the stock;
- Specifying the number of stocks;

- Saving the transaction.

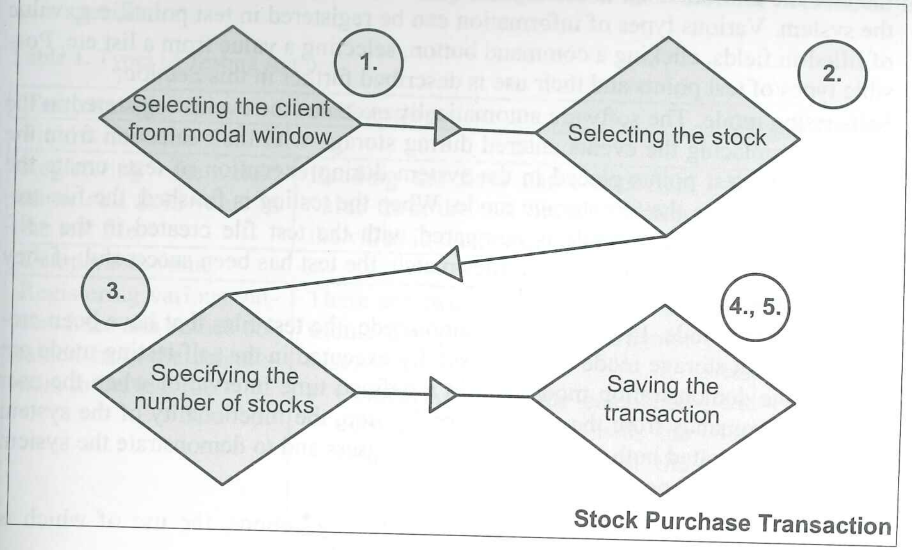

Fig. 3. Stock Purchase Transaction Process

To implement self-testing in the stock purchase transaction process, the system would have the following five test points, which various testing actions are written to:

1. Test point *Modal window* registers the client selected in it in the test storage file.
2. Test point *Field with value* registers in the test storage file the security specified for the transaction.
3. Test point *Field with value* registers in the test storage file the quantity of securities specified for the transaction.
4. Test point *Application event* registers in the test storage file the event of clicking on the button Save.
5. Test point *SQL query result* registers in the test storage file the data saved in the database after clicking on the button Save.

Classification of test points is outlined in detail below in this Section.

When a stock purchase transaction test case is registered, each of the points in the test storage file registers information that is used to play back the test. When a stock purchase transaction test is plaid back, the self-testing software, step by step, reads from and executes the actions registered in the test file. When the actions specified in the test file are executed, a new test file is created. When all the actions have been executed, the test files are compared; they should match if the tests have been successful. If the files do not match, the user is able to identify in the testing software application the point (command) in the test file that has been executed with errors.

Test points are placed by the developers in the system to achieve that the critical functionality of the system is covered. Test points are used as follows:

- Test storage mode. When the user creates a new test, the specified information in the test file and obtained in testing is registered in the test points implemented in the system. Various types of information can be registered in test points, e.g. value of filled-in fields, clicking a command button, selecting a value from a list etc. Possible types of test points and their use is described further in this Section;
- Self-testing mode. The software automatically executes the events registered in the test files, replacing the events entered during storage with their selection from the test file. The test points placed in the system during execution of tests create the same test file as in the test storage mode. When the testing is finished, the file created in the test storage mode is compared with the test file created in the self-testing mode. If the contents of the files match, the test has been successful; if they do not match, the testing has failed;
- Demonstration mode. In the demonstration mode, the test files that have been created in the test storage mode and successfully executed in the self-testing mode are used. In the demonstration mode, within a defined time interval or when the user executes commands from the test file step by step, the functionality of the system can be demonstrated both to teach new system users and to demonstrate the system functionality to any potential its buyers.

Self-testing software employs the following testing actions, the use of which is shown in Table 1:

- Field with value. The action is required to register a field filling-in event;
- Comparable value. This test point is necessary to be able to register and compare values calculated in the system. The test point can be used when the application contains a field whose value is calculated considering the values of other fields, values of which are not saved in the database;
- MessageBox. This test point is required to be able to simulate the message box action, not actually calling the messages. This is necessary as not all technologies provide a possibility to press the message button with the help of the system during test execution;
- Modal window. This test point is required to be able to simulate the modal window action, not actually calling the modal windows. This is necessary as not all technologies provide a possibility to access during test execution, after calling the modal window, the window from which the modal window is called;
- SQL query result. This test point registers specific values that can be selected with an SQL query and that are compared in the test execution mode with the values selected in the test storage and registered in the test file. The SQL query test point can be used after data have been saved to compare the data saved in the database, the data saved when registering the test and the data saved when performing the test repeatedly;
- Application event. This test point is required to register any events performed in the application, e.g. clicking on the button Save;
- Test execution criterion. This test point controls whether it is possible to execute the test. By using test execution criteria test points, it is possible to specify the criteria for the execution of the stored test. In the system self-testing mode, the test execution criteria points check whether the conditions specified in the test points

are fulfilled. If the criterion is not fulfilled, the test has failed and the user can access a detailed description of test execution, in which the reason for non-execution is specified.

Table 1. Types of Testing Actions

Test storage	Self-testing	Demonstration
Field with value		
Registering the field name and field value in the test file.	Reading the field name and field value from the test file and writing the value in the respective field.	See text execution mode.
Comparable value		
Registering various values, inter alia values obtained from the calculation, in the test file.	There are two cases (additional attribute that points to the particular case): • Comparing the value calculated during test execution with the value registered in the test file during test storage; • Using the value registered in the test file during test storage in test execution, not recalculating the value.	Using the value registered in the test file during test storage in test execution, not recalculating the value.
Message box		
Registering in the test file the message call and the action performed by the user.	The message is not shown to the user. Tests are executed taking into account the action performed by the user and registered in the test file.	In this mode, the message box is shown on the screen.
Modal window		
Registering in the test file the modal window call and the return values.	The modal window is not opened during test execution, and the modal window values registered in the storage mode are used from the test file.	In this mode, the modal window is shown on the screen.
SQL query result		
Registering the test results in the test file. Test results can be both a particular field value and a query result.	Comparing the values obtained in the result of test execution with the values registered in the test file during test storage mode.	Test points are not used in this mode.
Application event		
Registering application events in the test file.	Reading application events from the test file and executing them.	See text execution mode.
Test execution criterion		
The action is not used in this mode.	Controlling whether it is possible to execute the tests in the current state of database.	See text execution mode.

5 Implementation of Self-Testing

Key components of self-testing software are:

1. Test control block, which provides the following key functions:

- Selecting the test execution mode (execution of all or selected tests);
- Selecting the test mode. The user can specify whether tests should be executed in visible or invisible mode. The visible mode is intended for demonstrations; but if the user wants, they can follow test execution step by step. The invisible mode provides for faster test execution;
- Information on test execution. If the test fails, the control block will provide the user with information on reasons for the failure;
- Deleting tests and test files.

As of writing this paper, the first version of the test control block has been developed (Fig. 4. Test Control Block). The test control block has been developed with additional functionality and improved with user interface.

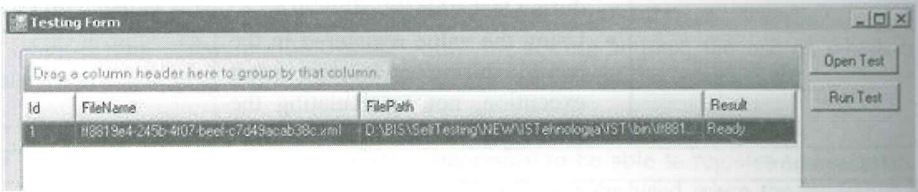

Fig. 4. Test Control Block

2. Library of test actions. The library contains the test action functions described herein. Testing action function calls are implemented in the tested system. Test functions are assigned parameters that characterise the test action. Testing functions, on the basis of the received parameters, make the respective records in the test file.
3. Test file (XML file). Test functions in XML file, using a particular structure, register the values that characterise the test case. The XML file structure consists of the following elements and their attributes:

- Form. Its attributes are the form name and the test point number, and its elements are test points;
- Action. The element is a set of other elements. It contains all the test points described in Section 4;
- Control. The element contains data on the control used in the test point. The element contains the following attributes: test point number, control name, event (e.g. change of value) called at the test point, control type;
- Value. Value element. Contains information on the value selected/entered by the control. In addition, the element can contain the value data type (e.g.: xsi:type="xsd:string" – which means that the control value is a string of symbols);
- Values. Element values. If the control contains a number of selected values, they are shown under this element. The element contains the Value element;

- Function. This test point determines whether a function has been called. The element contains the Parameters element and the following attributes: test point number, function name;
- Parameter. Function parameter. Contains information on the value of the parameter of the called function. In addition, the element can contain the value data type;
- Parameters. The element is a set of function parameters. The element can contain the Parameter elements;
- ModalFields. The element contains information on the return values of the modal window. The element contains the Fields element and the test point number attribute;
- Field. Modal window return value element;
- Query. The element contains the SQL query, which is executed by the system in the database. Its attribute is the test point number;
- ComparableField. The element contains information on the field that must be registered when the test is recorded in order to be able, when the test is played back, to check whether the value matches the value that was registered when the test was recorded. The element contains the following attributes: test point number, field name. The element contains the Value element, in which the field value is specified;
- DialogResult. The element contains information on the return values of the dialog box. The element contains the following attributes: test point number, dialog result;
- ChildForm. ChildForm matches with the Form element (form child). The element is required if another form is called from the form.

The system login test example, in which test points are located, is shown in the figure below (Fig. 5. System Login Test Example).

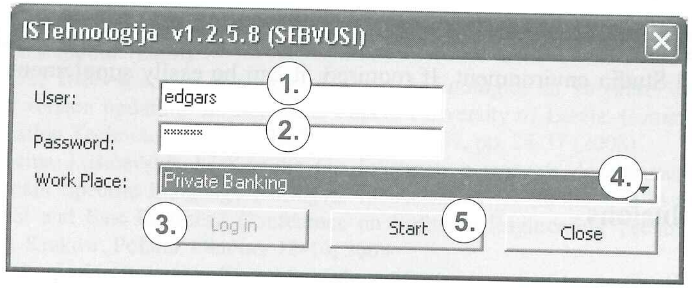

Fig. 5. System Login Test Example

The user performs the following actions in login form:

1. Entering the user name.
2. Entering the user password.
3. Clicking on the button Log in.
4. Choosing work place.
5. Clicking on the button Start.

When a system login test case is registered, each of the points in the test storage file registers information (Fig. 6. Test File Example) that is used to play back the test.

```
- <Actions>
 - <Control Id="1" Name="Username" Event="Username_TextChanged"
     ControlType="System.Windows.Forms.TextBox">
     <Value xsi:type="xsd:string">edgars</Value>
   </Control>
 - <Control Id="2" Name="Passwd" Event="Passwd_TextChanged"
     ControlType="System.Windows.Forms.TextBox">
     <Value xsi:type="xsd:string">123456</Value>
   </Control>
   <Control Id="3" Name="btnAuth" Event="btnAuth_Click"
     ControlType="System.Windows.Forms.Button" />
 - <Query Id="4">
     <Query>SELECT d.DVI_NOSAUK,d.DVI_ISN FROM IST_DVI d, IST_SLD s
       WHERE s.DVI_ISN=d.DVI_ISN AND s.DAR_ISN=72 order by 1</Query>
   </Query>
 - <Control Id="5" Name="Workplace"
     Event="Workplace_SelectedIndexChanged"
     ControlType="System.Windows.Forms.ComboBox">
     <Value xsi:type="xsd:int">20</Value>
   </Control>
   <Control Id="6" Name="StartWork" Event="StartWork_Click"
     ControlType="System.Windows.Forms.Button" />
</Actions>
```

Fig. 6. Test File Example

In the example (Fig. 5. System Login Test Example) the order in which test is recorded (1-2-3-4-5) is not fixed. The user in login form could perform actions in any order (for example 2-1-3-4-5). The order in which test case will executed will be the same as the sequence in which the test is recorded.

The test functions library described above can be used in projects developed in the MS Visual Studio environment. If required, it can be easily supplemented with new functions.

6 Conclusions

In order to present advantages of self-testing, the self-testing features are integrated in a large and complex financial system. Although efforts are ongoing, the following conclusions can be drawn from the experience:

1. Introduction of a self-testing functionality is more useful in incremental development model, especially gradually developed systems and systems with long-term maintenance and less useful in the linear development model.
2. Self-testing significantly saves time required for repeated testing (regression) of the existing functionality. This is critical for large systems, where minor modifications can cause fatal errors and impact system's usability.
3. Self-testing requires additional efforts to integrate the functionality of self-testing into software, to develop critical functionality tests and testing procedures.

4. The introduction of self-testing functionality would lower maintenance costs and ensure high quality of the system.
5. Self-testing does not replace traditional testing of software; it modifies the testing process by increasing significantly the role of developer in software testing.
6. Test points make test recording and automatic execution much easier. Test points ensure that tests can be recorded in a convenient and easy-to-read manner.
7. Test execution criteria test point determines the possibility to execute the test using the available data set.
8. If test execution criteria test points are used, it is not necessary to maintain the data set which was used to register the test.
9. If test points are used, the user can, independently from the developer, register and then repeatedly execute test cases.
10. Test execution criteria test point provides a possibility to execute tests in random order.

References

1. Bičevska, Z., Bičevskis, J.: Smart Technologies in Software Life Cycle. In: Münch, J., Abrahamsson, P. (eds.) Product-Focused Software Process Improvement. 8th International Conference, PROFES 2007, Riga, Latvia, July 2-4, 2007, LNCS, vol. 4589, pp. 262-272. Springer-Verlag, Berlin Heidelberg (2007)
2. Rauhvargers, K., Bicevskis, J.: Environment Testing Enabled Software - a Step Towards Execution Context Awareness. In: Hele-Mai Haav, Ahto Kalja (eds.) Databases and Information Systems, Selected Papers from the 8th International Baltic Conference, IOS Press vol. 187, pp. 169-179 (2009)
3. Rauhvargers, K.: On the Implementation of a Meta-data Driven Self Testing Model. In: Hruška, T., Madeyski, L., Ochodek, M. (eds.) Software Engineering Techniques in Progress, Brno, Czech Republic (2008).
4. Bičevska, Z., Bičevskis, J.: Applying of smart technologies in software development: Automated version updating. In: Scientific Papers University of Latvia, Computer Science and Information Technologies, vol .733, ISSN 1407-2157, pp. 24-37 (2008)
5. Ceriņa-Bērziņa J.,Bičevskis J., Karnītis Ģ.: Information systems development based on visual Domain Specific Language BiLingva. In: Accepted for publication in the 4th IFIP TC2 Central and East European Conference on Software Engineering Techniques (CEE-SET 2009), Krakow, Poland, Oktober 12-14, 2009
6. Ganek, A. G., Corbi, T. A.: The dawning of the autonomic computing era. In: IBM Systems Journal, vol. 42, no. 1, pp. 5-18 (2003)
7. Sterritt, R., Bustard, D.: Towards an autonomic computing environment. In: 14th International Workshop on Database and Expert Systems Applications (DEXA 2003), 2003. Proceedings, pp. 694 - 698 (2003)
8. Lightstone, S.: Foundations of Autonomic Computing Development. In: Proceedings of the Fourth IEEE international Workshop on Engineering of Autonomic and Autonomous Systems, pp. 163-171 (2007)
9. Bicevska, Z.: Applying Smart Technologies: Evaluation of Effectiveness. In: Conference Proceedings of the 2nd International Multi-Conference on Engineering and Technological Innovation (IMETI 2009), Orlando, Florida, USA, July 10-13, 2009
10. J. Barzdins, A. Zarins, K. Cerans, M. Grasmanis, A. Kalnins, E. Rencis, L.Lace, R. Liepins, A. Sprogis, A.Zarins.: Domain Specific languages for Business Process Managment: a Case Study Proceedings of DSM'09 Workshop of OOPSLA 2009, Orlando, USA

11. Diebelis, E., Takeris, V., Bičevskis, J.: Self-testing - new approach to software quality assurance. In: Proceedings of the 13th East-European Conference on Advances in Databases and Information Systems (ADBIS 2009), pp. 62-77. Riga, Latvia, September 7-10, 2009
12. Bičevska, Z., Bičevskis, J.: Applying Self-Testing: Advantages and Limitations. In: Hele-Mai Haav, Ahto Kalja (eds.) Databases and Information Systems, Selected Papers from the 8th International Baltic Conference, IOS Press vol. 187, pp. 192-202 (2009)