2013 20th Asia-Pacific Software Engineering Conference

Scrum Anti-patterns – An Empirical Study

Veli-Pekka Eloranta, Kai Koskimies,Tommi Mikkonen and Jyri Vuorinen
Department of Pervasive Computing,
Tampere University of Technology
Tampere, Finland
{firstname.lastname}@tut.fi

***Abstract*—Wide-spread adoption of the agile movement has rapidly changed the landscape of software industry. In particular, Scrum is an agile process framework that has become extremely popular in industry. However, the practical implementation of Scrum in companies rarely follows the text book ideals. Typically, companies deviate from the proposed Scrum practices for different reasons. While some deviations may be well motivated and reasonable, companies are often tempted to adjust Scrum for the company without clearly understanding the consequences of the deviations. In this paper our aim is to identify ways of potentially harmful mishandling of Scrum in industry based on empirical data collected in a survey. The identified (mal)practices are presented in a semi-formal manner as anti-patterns. The study resulted in 10 anti-patterns that express the context of the deviation, the deviation itself, the broken core principles of Scrum, and the possible consequences of the deviation.**

Keywords—Agile software development, Scrum, anti-patterns, Scrum problem areas

I. INTRODUCTION

The wide-spread adoption of the agile movement – culminating in many ways in the Agile Manifesto published in 2001 [1] at the level of ideals – has taken off in the software industry in a rapid pace. Within a relatively short period, an increasing number of organizations have adopted an agile way of working in their R&D departments. However, when inspected more closely and in more detail, by phrase "we are agile in software development", companies often mean simply, "we are using some practices or ideas of Scrum". This is often referred as ScrumBut [14].

Scrum is a simple, iterative framework for project management [12, 13, 15]. At present, the adoption of Scrum has progressed in several organizations so far that it is possible to start collecting empirical data on the use of Scrum. To consolidate accurate, practical knowledge that is relevant to companies developing software, it is crucial to gather data on the Scrum practices in action for determining which parts of the methodology pose problems in industry. In this paper, our goal is to identify empirically the problem areas in Scrum and practices that are hard to adopt in the organizations, and to present the found Scrum pitfalls in a systematic manner as so-called anti-patterns [2]. Anti-patterns are common practices that are seen initially convenient and appropriate, but that are actually harmful in the long run and should therefore be avoided. An anti-pattern is often associated also with an alternative, recommended practice, which is more appropriate in most of the cases.

In principle, there are two kinds of problems in adopting Scrum: (1) harmful deviations from recommended Scrum practices and (2) recommended Scrum practices that are for some reason unsuitable in a particular context. Both types of problems are important to be recognized. For the first type, a company starting to use Scrum should be aware of common deviations that may look reasonable but that are actually harmful. For the second type, understanding well-motivated deviations from the standard, text-book Scrum provides information for refining the methodology to fit the purposes of companies developing software. In this paper we will not distinguish between these two types of Scrum problems. The rationale for this decision is that even though a Scrum deviation may be well-motivated in a particular context, the deviation has harmful consequences in most cases. Instead, we discuss in each anti-pattern the possible exceptions when the Scrum deviation might be sensible. In general, the alternative "good" practice which is part of an anti-pattern is the recommended text book Scrum practice.

To mine typical Scrum pitfalls in industry, we have conducted a survey of Scrum related practices in IT companies that use Scrum. The results of the survey are presented in this paper in terms of basic Scrum practices. For each basic practice, we summarize the contents of the interviews, especially concentrating on the ways the practices have been realized in the companies and possible deviations from Scrum recommendations. Then, we formulate the identified deviations from Scrum as anti-patterns using a format specifically designed for this purpose. These anti-patterns, together with the empirical data supporting them, are the main contribution of this paper.

The rest of this paper is structured as follows. Section II provides the basic information concerning the survey method. In Section III we summarize the results of the survey structured according to the basic Scrum concepts, discussing the extent to which companies follow the related Scrum practice and the possible deviations. Section IV condenses the contribution of this paper by formulating the identified problematic practices as anti-patterns. Section V discusses the limitations of this study, and Section VI summarizes previous work on the usage of Scrum in industry, with comparisons to our work. Towards the end of the paper, Section VII draws final conclusions and introduces directions for future work. We assume familiarity with Scrum throughout the paper.

1530-1362/13 $31.00 © 2013 IEEE
DOI 10.1109/APSEC.2013.72

II. SURVEY METHOD

Survey Research [6], commonly used to assess opinions and feelings, was selected as a research method for this study. The method gives an opportunity to identify characteristics of a group of individuals, and based on the results, one can compare the views of different populations.

The interview covered 11 IT companies in Tampere region, which is one of the centers of IT industry in Finland. The representatives of selected companies had participated in a certified Scrum master course. The survey is based on structured interviews of persons that play (or have played at the time) a major role in software development organizations using Scrum. Companies participating in the survey are of mixed background, some working for customer projects, and some acting in a more product-driven fashion. The range of systems includes embedded devices as well as more traditional applications, such as web-based software.

In total, 18 representatives of teams were interviewed, including project, development and quality managers as well as developers. A typical title of a person being interviewed for the survey was company development manager, who can provide as wide and covering information regarding company's software development process as possible. In addition, since the majority of the interviewed persons had senior positions, they could also provide information on how Scrum was integrated with other activities of the company.

There were 48 questions in the survey. They were composed so that the answers would cover all the objectives of this study. In the interview, additional questions were asked concerning Scrum basics whenever necessary to find out if the organization has deviated from the practice and adopted their own way of working or if the organization had not taken the practice in use in the first place. The complete list of questions can be found at [7].

III. SURVEY RESULTS

A. Sprints

Sprint is a time-boxed period where the actual work in Scrum takes place. The original Scrum paper [12] does not define any length for the sprint, but among the practitioners the length of 2 to 4 weeks has become a de facto practice. Scrum guide [15] also advices to use 2 to 4 week sprints. The results of the question on sprint length are shown in Table I.

Table I. Sprint lengths in the interviewed teams

< 2 weeks	2 weeks	3 weeks	4 weeks	> 4 weeks	Varying length	No sprints
1	9	3	2	1	1	1

As Table I indicates the most widely used sprint length is 2 weeks. One company reported having a varying sprint length meaning that if the work takes more time than planned then the sprint is extended. Furthermore, one company reported that they did not use sprints at all, even though they claimed that they used Scrum.

Most of the teams had experimented with the sprint length. The most typical scenario was that the team had started with 4 week sprints and noticed that it is too long. In most such cases 4 weeks was too long as the team could not respond to customer requests and the feedback loop became too long. In addition, two teams reported that when they had used 4 week sprints, only two thirds of the sprint's work was done. When they shortened the sprint length, they could achieve situation where all tasks were completed at the end of the sprint.

Roughly half of the teams had tried sprint lengths shorter than two weeks, most typically 1 week or 1.5 weeks. However, they had abandoned short sprints as the management overhead became too large. Sprint planning and sprint reviews was taking too much of the sprint length. Furthermore, typically two weeks were short enough time to be able to satisfy customer's change requests.

One team had two overlapping iterations at the same time. They had iterations of 4 weeks and 2 week sprints within the iteration. They had to take this approach as the customer did not want to participate in meetings every two weeks. Some of the teams had sprint length of 2 to 4 weeks, but they had separate implementation and testing sprints meaning that every other sprint they implemented new features and the next sprint was for testing those features. Of course, this approach does not produce potentially shippable product increment each sprint.

B. Testing

In Scrum, every sprint should produce a potentially shippable product increment. In order to accomplish that, the software has to be tested before the sprint ends. Typically this is incorporated in the definition of "done". In the interview, we asked the companies to what extent the software is tested within a sprint. Results are shown in Table II.

Table II. State of the testing of software after each sprint

Not tested	Unit tested	Integr. tested	System tested	Acceptance tested	Delivered and installed
3	8	2	2	2	1

Eight teams had the software unit tested at the end of the sprint. However, two of those teams said that they had efforts going on to make the testing more automated and the goal was to achieve automated system testing. In some cases, the system testing was carried out always in the next sprint by a separate team. This is, however, problematic as new functionality might be already written on top of the old code which is still in the testing phase. Only three teams had the code acceptance tested within the sprint. One of these teams was also deploying the product within the sprint. However, they had had to limit the deployment to every other sprint as the customer did not want to update the software so often.

According to the interviewed teams, the biggest obstacle on the road to test the code more within the sprint was "waterfallish" thinking where testing is carried out at the end

of the sprint and the lack of automation in the testing. In addition, some of the interviewed companies were manufacturers of large machines and in these systems testing is even harder as one should test the software on the hardware before it can be shipped.

C. Specifications

In Scrum it is advised to document requirements as user stories or use cases in the product backlog. Table III shows the methods interviewed companies used to make specifications for their products.

Table III. Specifications documentation

Big requirement documents	User stories	Use cases	Face-to-face	Feature descriptions
4	8	3	1	2

As the results show, four teams were still using traditional big requirement documents for specifications. Most of the teams (8) were using user stories. Two of these teams reported that they have separate UI specifications or more detailed specifications when necessary to accompany user stories. Three teams used use cases. One of these teams said that they prefer use cases over user stories as use cases document context for the feature and provide some rationale for the functionality. One team had face-to-face discussions where the product was specified. After discussions with the customer and the product owner they documented just enough specifications that they needed to remember what was discussed. Two teams used only short feature descriptions in their product backlog.

As stated by Cockburn [3], the most efficient way of communicating is face-to-face. It would be advisable to have face-to-face discussions especially between the customer and the development team when making the specifications. If the requirements are communicated using written user stories, the communication is one-directional and not efficient as the development team cannot ask questions from the customer.

D. Product owner

Product owner is responsible for maximizing the value of the product [15]. Product owner is also responsible for organizing the product backlog and creating enabling specifications for the product owner team. Table IV shows the distribution of the different approaches with respect to the product owner.

Surprisingly, altogether 7 teams reported not to have a product owner at all. Furthermore, 3 teams had a product owner who was customer representative. This is a bit alarming as the product owner is responsible for optimizing ROI. If the customer is in a position to optimize ROI for the company, it cannot be beneficial for the company in the long run. One team reported to have a shared product owner. This means that the company had one product owner who was managing more than one product. It was not very surprising that this team had problems as the product owner did not have enough time to create enabling specifications.

Table IV. Product owner role

No product owner	Own product owner	Product owner from customer	Shared product owner
7	7	3	1

One team also reported to have product owner team, i.e. there are multiple product owners that work as a team. They had evolved to this practise from the single product owner as the teams had been complaining that the single product owner did not have enough time to address the questions the teams had.

One interesting point that was discovered multiple times in the interviews was that teams who did not have product owner and were working on customer projects said that having a product owner would not work for them. They argued that if they were building their own product, it might work better. On the other hand, multiple teams building their own product and having no product owner reported that the product owner role would work for them only if they were doing customer projects. So independently of the project type, the teams found the product owner role to be problematic.

E. Product backlog

Product backlog is an ordered list of things that might be needed in the product and acts as a single source of requirements for the product [15]. The product backlog should be ordered so that the most valuable, highest-risk or priority items are on the top (taking dependencies into account). These top items are also the most analyzed, clear and ready for development. Table V shows the usage of product backlog in the interviewed teams.

Table V. Usage of Product backlog

No Product backlog	Product backlog	Ordered product backlog	Multiple product backlogs
3	7	7	1

As can be seen from Table V, most of the teams were using product backlog. Only three teams were not using it. None of these three teams reported that they have evolved away from this practise; they just had not adopted it. The largest observed problem with the product backlog was that in roughly half of the teams, the product backlog was not ordered. When considering the results of the product owner questions (7 teams not having a product owner) is not a big surprise that the product backlog was not ordered in so many companies. The interviewed teams reported that the main obstacle in ordering the backlog was that the product owner did not have enough time for it or product owner did not have required competence to order the list. However, in the interviews we also discovered that some teams were ordering the product backlog with the product owner as he was not able to do it alone. Only one team reported that they work for more than one product owner and select items from multiple backlogs to the sprint.

F. Estimates

Work estimation is an important aspect of Scrum, enabling the efficient use of resources. In the survey we asked who is responsible for making such estimates, if anyone. We also investigated how many of the teams used hours for estimation and how many used so called estimation points. The results concerning the responsible party of the estimates are shown in Table VI.

Table VI. Who estimates the PBIs

No estimates	Project manager / PO estimates	Team estimates
2	6	10

Two of the interviewed teams did not do estimates at all. For 6 teams, a project manager or a product owner made the estimates. This number is surprisingly big and indicates that the project manager or the product owner dictates which items are implemented in which sprint. Later on the question about the self-organization confirmed this conclusion. In total, ten teams produced the work estimates by themselves.

Out of the 16 teams having estimates, eight created their estimates using hours. On the other hand, eight used estimation points or other imaginary unit to estimate the work amount. Planning poker was used by five teams out of the eight applying estimation points.

G. Sprint burn-down charts and velocity

Sprint burn-downs are used to illustrate remaining work in sprint or in some cases in release [15]. Generally the idea is to burn the chart down when a task is completed. However, different variations are often used.

In total seven teams did not use burn-downs whereas eleven teams used them. However, in interviews one team said that they have it in use, but they have never seen it. The project manager was drawing it for himself to get estimates concerning the possible failure of the sprint. Seven teams did not use burn-downs. The main reason for not using them was that they did not see any benefits gained out of drawing the chart – it is just extra work to maintain the chart. For some teams a reason was just that there was no suitable place in the wall for the chart.

Out of the eleven teams who used burn-down charts, eight had partial burn-downs meaning that they used hours in the burn-down chart and once they had worked on some task, they immediately removed it from the remaining work. Only three teams burned down the graph when the task was completed.

Additionally, we asked if teams know their velocity, meaning how many work items they can do within a sprint. Eight teams used velocity to estimate the amount of work items they can deliver in sprints. Ten teams did not know their velocity.

H. Team disruption

Generally, Scrum advices to protect the team from outside disruptions. However, in practice this might be hard to achieve as team members have to answer phone calls, take bug reports and maintain old projects. Therefore, it is not very surprising that 13 of the interviewed teams were disrupted during the sprint and only five teams were not. In one case, it was the project manager disrupting the team, in other cases, typical cause of disruption was the customer of the current (or old) project.

I. Self-organization and cross-functionality of the team

In Scrum, it is critical that teams are cross-functional meaning that they have all the necessary expertise to deliver working software. Furthermore, the team should be self-organized so that each team member can choose what to work on and there is no one assigning tasks. Therefore, we asked if the companies had cross-functional teams and are they self-organizing. Ten teams had the necessary skills (e.g. design, coding and testing) to deliver a working product. In eight cases there were, for example, separate testing teams and therefore the team was not cross-functional. This usually causes problems, a typical one being that the product is not ready to be delivered after the sprint as it still needs to be tested. Furthermore, testing typically takes place during the next sprint and, consequently, new code has often been written on top of the code being tested at the same time.

The last question about the Scrum practices concerned the self-organization of the team, that is, to what extent the team gets to decide what they are working on. Half of the interviewed teams, i.e. nine, were self-organized and the other half had a project manager to assign tasks to certain persons.

IV. Identified Scrum Anti-patterns

A. Anti-pattern format

We will use the format presented in Table VII for the presentation of the anti-patterns. The used anti-pattern format is derived from [2] with small modifications to better suit the organizational nature of the presented anti-patterns. The Scrum recommendation is taken from Scrum literature, for example [15]. The context is extracted from the results of the interview, reflecting the explanations the interviewees gave for the Scrum deviations. The consequences are partly taken from the interviews as the problems noted by the interviewees and partly from Scrum literature, mostly [15]. Often the existence of another anti-pattern was observed to be one of the causes of an anti-pattern. Exceptions are situations showing up in the interviews that were considered to be "justified Scrum deviations" by the authors. In some cases, no notable exceptions could be identified.

B. List of identified anti-patterns

Ten anti-patterns were identified based on the survey, as presented in Table VIII. The following criteria were used to accept a practice as an anti-pattern: 1) the practice is a deviation of Scrum recommendations, 2) the practice was observed in more than one team, and 3) the practice has

unfavourable consequences noted explicitly either by the interviewed team or by Scrum literature.

Table VII. Anti-pattern format

Anti-pattern field	Contents
Name	The name of the anti-pattern
Scrum recommendation	The related Scrum recommendation
Context	Specific conditions under which the anti-pattern typically occurs
Solution	The anti-pattern (usually harmful) solution or Scrum deviation
Consequences	The consequences of using the anti-pattern
Exceptions	Possible situations in which the use of the anti-pattern could be justified

Table VIII. Identified anti-patterns

Name: **Too long sprint** *Scrum recommendation*: 2 weeks sprints. *Context*: Early experimenting with Scrum, waterfall legacy of long production cycles, customers unwilling to participate in short intervals. *Solution*: Using 4 weeks or even longer sprints. *Consequences*: In spite of longer working time for sprints, the planned tasks tend to be unfinished at the end of sprint, possibly because the first weeks are not used efficiently enough and too large tasks are allocated to the sprints. Consequently diminishes team commitment to deliver the items at the end of the sprint. Feedback cycle becomes so long that some of the work might not be needed anymore. *Exceptions*: If the development must be synchronized with external work that has slower pace (e.g. hardware development), longer sprints may be justified.
Name: **Testing in next sprint** *Scrum recommendation*: All testing of software is done within the sprint that creates the software, to enable the shipping of a working product increment. *Context*: Waterfallish thinking of the team, separate testing team, lack of testing automation, too short sprint anti-pattern, too long sprint anti-pattern. *Solution*: Testing is done in the next sprint following a development sprint. *Consequences*: New code may be written on top of non-tested code. The product is not always in a "potentially shippable" state. Bugs need to be fixed long after the code is written. As the bug is found after several weeks from implementing the feature, it might require some time to recall how the feature was implemented. Increases cognitive load. *Exceptions*: In an embedded system, it may be necessary to test the code together with real hardware, which might be available only at the later stage of the development.
Name: **Big requirements documentation** *Scrum recommendation*: Product backlog contains all potential items that are going to be implemented and this list is ordered by value, risk, or dependencies, for instance. Items might be described as user stories or use cases and so on. *Context*: Waterfallish thinking of the team, legacy company guidelines for requirements documentation. *Solution*: Traditional big requirements documentation is produced prior to the development sprints. *Consequences*: Requirements are not well understood by the team. Requirements are not ordered by the value or other criteria. Thus, it is harder to decide which items should be implemented next. *Exceptions*: In safety-critical systems, various regulations enforce the existence of large requirements documentation and traceability to the code.
Name: **Customer product owner** *Scrum recommendation*: Product owner role is necessary to ensure that the product produces value to the customer. *Context*: Insufficient understanding of the role of a product owner, lack of persons suitable for that role, customer does not trust the development team. *Solution*: A customer representative acting as a product owner. *Consequences*: Requirements are not well understood by the team, leads to Customer caused disruption anti-pattern Customer tries to add new items to the sprint during the sprint. Customer optimizing the return on investment for the company. *Exceptions*: If suitable person to represent customer is not available, Product owner surrogate could be used temporarily.
Name: **Product owner without authority** *Scrum recommendation*: Product owner is responsible for managing Product backlog and deciding which items maximize the value of the product. *Context*: Insufficient understanding of the role of a product owner, suitable persons are not motivated to use Scrum, general lack of interest, large organizations where responsibility over products and development may be fragmented. *Solution*: A development team member is promoted as product owner as product managers are not interested in using Scrum. *Consequences*: Product owner does not have the authority to decide on which items are implemented and which are discarded. Product owner can not really make decisions concerning the product. There might be other managers to above product owner to make these decisions. Product owner is not necessarily in direct contact with the customer. *Exceptions*: If product owner with authority is not temporarily available, a surrogate product owner could be used for a couple of sprints.

Name: **Unordered product backlog**
Scrum recommendation: Product backlog is ordered to give precedence for high-risk or the most valuable items.
Context: Product owner without authority anti-pattern, Customer product owner anti-pattern, insufficient competence of the product owner.
Solution: Product backlog is not ordered, but teams select items based on their own judgement.
Consequences: Lacking vision of the risky or valuable elements of the product. Building wrong features. Only the nice features get implemented. Features being hard to implement or test are left to the backlog.
Exceptions: Projects where the requirements are fixed and given up-front by the customer. In addition these requirements are not going to change. Some initiatives by governments and military might have such requirements.

Name: **Work estimates given to teams**
Scrum recommendation: Teams should produce work estimates for themselves.
Context: Hierarchical working practices of the company. Need to know an estimate in advance on how much the product will cost.
Solution: Product manager or product owner produces work estimates.
Consequences: Lacking commitment of the team. Wrong estimates: team is better to estimate than a single person who is not going to implement the task. Out-dated estimates as items have dependencies.
Exceptions:

Name: **Invisible progress**
Scrum recommendation: The progress of the work should be made visible with burn-down charts.
Context: Hierarchical working practices of the company, unsuitably distributed team
Solution: Product manager produces burn-down chart for herself, or no visual representation of the progress is produced. Sometimes partially completed tasks are marked as finished.
Consequences: Lacking progress awareness of the team, risk of failing to deliver all features in a sprint. Problems can be hidden. Scrum Master's work becomes harder.
Exceptions: Very small projects where people know each other, share a room, and can discuss the progress.

Name: **Customer caused disruption**
Scrum recommendation: Teams should be protected against any outside disruptions.
Context: Improper product owner anti-pattern, close and interactive co-operation with a customer.
Solution: Customers are allowed to communicate directly with team members and they can negotiate with the team to add new features to the sprint
Consequences: Work flow of the team is interrupted, reducing the efficiency of the team. New features may crawl in to the product without Product owner's involvement. The value stream is compromised.
Exceptions:

Name: **Semi-functional teams**
Scrum recommendation: Teams should be cross-functional, providing all the necessary skills to carry out the full realization of shippable products.
Context: Testing in next sprint anti-pattern, lacking expertise in the company staff, division to different teams according to the expertise.
Solution: Team produces only certain aspect of the shippable product.
Consequences: Divided responsibility of producing a shippable product, less committed team. Extraneous need for cross-team communication. Slower feedback loops.
Exceptions: Highly specialized and demanding software architecture design carried out before the sprints.

C. *Scrum anti-pattern usage and company characteristics*

To investigate the possible effect of the characteristics of the company on the tendency to resort to Scrum anti-patterns, we divided the companies into three categories according to their size: small (less than 30 developers), medium (30-100 developers), and large (over 100 developers). In addition, the companies were also categorized according to years of Scrum experience: less than a year, 1 to 2 years, and over 2 years. Table IX shows the results.

Two significant observations can be made from the results. First, it seems that smaller companies follow Scrum principles more closely, avoiding anti-patterns. This might be due to fact that large organizations tend to have legacy practices which prevent further adoption of new practices. Lindvall et al. [8] also states that in large organizations, new practices may result in double work as old practices need to be followed when new practices are taken into use.

The second observation from the result is that companies having more Scrum experience tend to use anti-patterns more. On one hand this can be explained by the fact that companies might have changed their way of working as a part of continuous improvement. On the other hand, if this is the case, they have adopted anti-patterns as a part of process improvement, which, of course, is not the intention. This might need further research to find out the reasons behind the anti-pattern adoption in these companies and what kind of consequences that might have.

Two anti-patterns seem to be common despite of company size and Scrum experience: Testing in the next sprint and Customer caused disruption. The popularity of the first anti-pattern indicates that the transformation to agile methods is particularly challenging in testing. The frequency of the second anti-pattern hints that companies do not want that the used development process affects their customer relationships and therefore let the customer disrupt the development teams.

V. LIMITATIONS

The survey was carried out in a limited region, which may influence the results. However, as different kinds of companies were included in the survey, we believe that the results can be generalized. In Survey Research, the question

design is critical to get useful and valid data [6]. In this survey, the main questions, however, are directly related to Scrum basic concepts and if the interviewee understand Scrum, the questions are likely to produce useful data. Therefore, question design is not a major threat to the validity of this survey.

Table IX. Summary of Scrum anti-patterns in different companies

Name	Company size			Scrum Experience		
	S	M	L	<1	<2	2+
Too long sprint		2	2	1		3
Testing in next sprint	1	1	4	1	1	4
Big requirements document		1	3	1		3
Customer product owner	1	1	2		2	2
Product owner without authority		2	3	1		4
Unordered product backlog	1	1	3	1	1	3
Work estimates given to teams		1	1	1		1
Invisible progress	1		3		1	3
Customer caused disruption	2	2	4	1	3	4
Semi-functional teams		1	2	1		2

Another limiting factor is that the number of interviewed companies was relatively small, which might lead to biased sample [6]. Conducting this kind of interview study for a large number of companies would be difficult, not only because of the required amount of resources but also because companies are not necessarily willing to share information about their problems, and even use some critical resources to do it.

Despite the above limitations, the results of this survey are in line with other survey-based studies, as will be discussed next. Still, it is likely that other anti-patterns would be probably found in other surveys, which does not invalidate the findings here.

VI. Related Work

As far as we know, there is no prior empirical work on identifying Scrum anti-patterns. In contrast, numerous positive properties have been associated with agile development approaches. Team communication – giving and receiving frequent feedback – is often reported to have increased when adopting agile methods in a company (see e.g. [10]). Also the regular Team meetings have been reported to support the collaboration and to give everyone better general view of the work progress. Customers are seen as valuable resources, and the customers themselves experience more active participation in the process – in contrast to the one of the weaknesses of the traditional waterfall approach where customers' current needs are not addressed until later on in the project [4]. Further benefits of agile approaches also include better process control, transparency, and improved product quality because of practices such as continuous integration and controllable-sized tasks [3]. These were confirmed in our interviews.

A recent study on agile methods by VersionOne gathered data from over 4000 survey participants. The study shows that Scrum (including also different kinds of Scrum hybrids) has 72% market share [16]. The study also lists the main benefits that companies reported being gained from the use of agile methods. The benefits include the following: 1) better control on managing changing priorities, 2) improved visibility and team morale, 3) quicker time-to-market and 4) increased productivity. The study also lists that the leading causes for failed agile projects are the following: 1) lack of experience in agile methods, 2) the organization culture does not support agile ways of working, and 3) external pressure to use traditional waterfall practices. All these problem areas were also identified in our interviews. In addition, the report lists barriers to further adoption of agile practices. The most significant barriers are: 1) lacking ability to change organization culture, 2) general resistance to change, 3) availability of skilled personnel, and 4) lack of management support. These all are also in line with our results from the interviews, giving further confidence towards the validity of our results. However since our results have been gathered with personal interviews, we believe the results have more depth than what can be collected with questionnaires only. In particular, the potential problems and deviations are easier to identify in personal interviews.

Another survey conducted by Salo et al. [11] gathered data from 35 projects from 13 software organizations from eight different European countries. The survey was carried out using web-based questionnaire, and the goal was to find out the level of adoption and the usefulness of agile methods such as Scrum and XP. The study was carried out in 2006, so the data is rather old. This can be seen in the results: Scrum was used only in 27% of companies who participated in the study. When compared to the survey by VersionOne [16], the market share differs radically. Furthermore, Salo et al. [11] report that the most popularly used Scrum practice was product backlog. However, our results imply that the situation has changed and the most widely used Scrum practice is constituted by sprints, arguably reflecting the fashion the actual work is organized. Moreover, also other practices that are intimately associated with actual development activities, such as the daily Scrum, were widely used in the companies participating in the study.

A summarizing comparison of different benefits and issues of agile methods has been given by Petersen [9]. In addition to Petersen's work, the benefits and challenges of agile methods have also been studied by Tore Dybå and Torgeir Dingsøyr [5]. Their systematic article reviews several virtues and benefits of these methods as well as the associated problems and challenges. In general, these results fall along the lines of our survey as well.

VII. CONCLUSIONS

Due to the rapid pace of new discoveries in the field of software engineering, new practices are commonly taken use before conclusive evidence on their results is available. It is only after a while when the new practices mature and get adapted to the actual needs of the different companies, and the problematic elements of the practices can be identified. At this point, it is possible to gather empirical evidence on the use of the practices.

In this paper, we have introduced the results from a series of interviews in software industry, where the goal was to understand how widely Scrum is deployed, and what are the most notable deviations from Scrum recommendations, as well as the rationale behind these deviations. The list of anti-patterns derived in this work gives a good picture of Scrum practices that are in general found difficult to follow by companies. This list of anti-patterns can be exploited by practitioners taking Scrum into use in a company, to pay attention to seemingly attractive Scrum deviations that may nevertheless have unpleasant consequences. The list can also be used by researchers to understand the problematic aspects of Scrum in real life, and to provide guidance and practice refinements to avoid resorting to these anti-patterns.

Since in this paper we only addressed basic Scrum practices on the development side, the results are inconclusive when considering the integration of Scrum practices in the activities of companies. To gain better confidence on the results, the survey should be conducted on a wider range of companies. Furthermore, issues such as synchronizing the activities of individual Scrum teams into a larger system e.g. in the form of Scrum of Scrums [13] were not considered. Constructing a survey on such issues forms an interesting piece of future work.

ACKNOWLEDGMENTS

This work is partly funded by the Finnish Funding Agency for Technology and Innovation in the context of project Sulava. The authors wish to thank all the interviewed companies for their cooperation.

REFERENCES

1 Agile Alliance. Manifesto for Agile Software, Available at http://agilemanifesto.org (2001)

2 Brown W.J. et al.: Antipatterns - Refactoring Software, Architectures, and Projects in Crisis. Wiley 1998

3 Cockburn, Alistair. Agile Software Development, 1st edition, December 2001, pages 256, Addison-Wesley Professional, ISBN 0-201-69969-9.

4 Coplien, James O. and Bjørnvig, Gertrud. Lean Software Architecture for Agile Software Development. August 2010, pages 376 , Wiley, ISBN 978-0-470-68420-7.

5 Dybå, Tore and Dingsøyr, Torgeir, Empirical studies of agile software development: A systematic review. Information and Software Technology, Vol 50, Number 9-10, 833-859 (2008)

6 Easterbrook, S., Signer, J. Storey, M-A. Damian, D. "Selecting Empirical Methods for Software Engineering Research" In: Guide to Advanced Empirical Software Engineering, Shull, F., Singer, J., Sjöberg, D. (eds), 2008, ISBN 978-1-84800-043-8

7 Eloranta Veli-Pekka. Interview questions. Available at http://www.cs.tut.fi/~elorantv/interview_questions.html (2012)

8 Lindvall, M.; Muthig, Dirk; Dagnino, A.; Wallin, C.; Stupperich, M.; Kiefer, D.; May, J.; Kahkonen, T., "Agile software development in large organizations," *Computer* , vol.37, no.12, pp.26,34, Dec. 2004

9 Petersen, K. and Wohlin, C., A comparison of issues and advantages in agile and incremental development between state of the art and an industrial case. Journal of Systems and Software, Vol. 82, Number 9, 1479-1490 (2009)

10 Pikkarainen, M, Haikara, J., Salo, O. Abrahamsson, P. Still, J., The impact of agile practices on communication in software development. Journal of Empirical Software Engineering, Vol. 13, number 3, pp. 303-337 (2008)

11 Salo, Outi and Abrahamsson, Pekka, Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. IET Software Vol. 2, Number 1, 58-64 (2008)

12 Schwaber, Ken, SCRUM development process. In: Proceedings of 10th Annual Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA), 1995, p. 117-134.

13 Sutherland, Jeff, Agile Can Scale: Inventing and Reinventing Scrum in Five Companies. Cutter IT Journal, Vol. 14, Issue 12, (2001).

14 Sutherland, Jeff, ScrumBut Test aka The Nokia Test. Available at http://jeffsutherland.com/scrumbutttest.pdf (2010)

15 Sutherland, Jeff and Schwaber, Ken, The Scrum Guide – The Definitive Guide to Scrum: The Rules of the Game. Available at http://www.scrum.org/storage/scrumguides/ Scrum%20Guide%20-%202011.pdf (2011)

16 VersionOne 7th annual state of agile survey. Available at http://www.versionone.com/pdf /7th-Annual-State-of-Agile-Development-Survey.pdf (2012)