Tampereen teknillinen yliopisto. Ohjelmistotekniikan laitos. Raportti 17
Tampere University of Technology. Department of Software Systems. Report 17

Veli-Pekka Eloranta & Kai Koskimies

Sulava Scrum Survey Report

TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Tampereen teknillinen yliopisto. Ohjelmistotekniikan laitos. Raportti 17
Tampere University of Technology. Department of Software Systems. Report 17

Veli-Pekka Eloranta & Kai Koskimies

Sulava Scrum Survey Report

Tampere University of Technology. Department of Software Systems
Tampere 2011

ISBN 978-952-15-2660-2 (printed)
ISBN 978-952-15-2661-9 (PDF)
ISSN 1797-836X

SUMMARY

Sulava Scrum Survey Results
Report, 27 pages, 3 appendix pages
September 2011

This document presents the results of interview survey on Scrum practices and practices of architecture work in agile development methods. Survey was carried out between September and December in 2010 and 12 companies participated in the survey.

The survey was carried out by interviewing architects, designer and managers of the companies. So called Nokia test was used as a part of interview questions to find out how agile companies are. In addition, Nokia test reveals much about the Scrum practices used in a company. Other questions of the interview concerned the architecture work in Scrum.

The results do not present the practices or agility of the whole company. The results are in many cases based on views of single design team or views of couple of teams. The main results include research on which Scrum practices are widely adopted and which are not. The results show that many companies are struggling with the Product Owner role. Product Owner role is in many cases the bottleneck as there are so many responsibilities for the Product Owner. Furthermore, sometimes the Product Owner role is completely neglected and the Scrum team is working directly with the customer or Product Owner is taken from the team and there is a long chain of managers between Product Owner and the customer.

The results also show that there are at least three way to carry out architecture design while using Scrum: Sprint 0, separated architecture process and creating the architecture in sprints by the team. These approaches have their own pros and cons and there is no one answer which of them should be used. One should inspect and adapt and choose which approach suits the best the situation at hand.

Table of Contents

Glossary

ADD Architecture Design Document is a document describing the software architecture of the system.

CSM Certified Scrum Master.

CSPO Certified Scrum Product Owner.

HIL Hardware-in-the-loop simulation. A technique that is used in the development and test of complex real-time embedded systems.

HW Abbreviation for hardware.

Nokia Test A simple way to determine if team has basic Scrum practices in place.

PB See Product Backlog.

Product Backlog The product backlog is the master list of all requirements (functionality and non-functional requirements) in the product. Items in this list are called PBIs.

PBI Product Backlog Item (PBI) is a work item in product backlog which typically are expressed as user stories, use cases and UI protos.

PO See Product Owner.

Product Owner Person who represents the stakeholders and the business in Scrum.

ROI Return on investment.

SW Abbreviation for software.

Scrum Master Person who maintains the Scrum process. Fights impediments in order to enable the team to perform better.

1 Introduction

This document presents the results of interview survey on Scrum practices and practices of architecture work in agile development methods. The survey was carried out between September and December in 2010 and 12 Finnish companies participated in the survey. Companies were mainly from the domain of embedded real-time control systems.

The survey was carried out by interviewing architects, designers and managers of the companies. So called Nokia test [1] was used as a part of interview questions to find out how agile companies are. Other questions of the interview concerned the architecture work in Scrum.

Section 2 introduces Scrum and its main concepts and practices. Section 3 summarizes the results of the Nokia test questions. Architecture work practices in Scrum are presented and discussed in Section 4. Furthermore, Scrum process best practices were also collected during the interviews. These best practices are presented in Section 5. Section 6 finally concludes this paper.

2 Scrum in a nutshell

This section and subsections summarize Scrum framework and its most important concepts. The scrum process is visualized in 1. Scrum uses two to four week iterations called sprints. Every sprint produces potentially deployable product or increment to the product. Product backlog is an ordered list of features containing every desired feature or change to the product. Product backlog items (PBI) are usually described with a user story or a use case. User interface sketches and other specifications are linked with the user stories or use cases as necessary. All stakeholders of the product can add or remove any PBIs whenever they want.

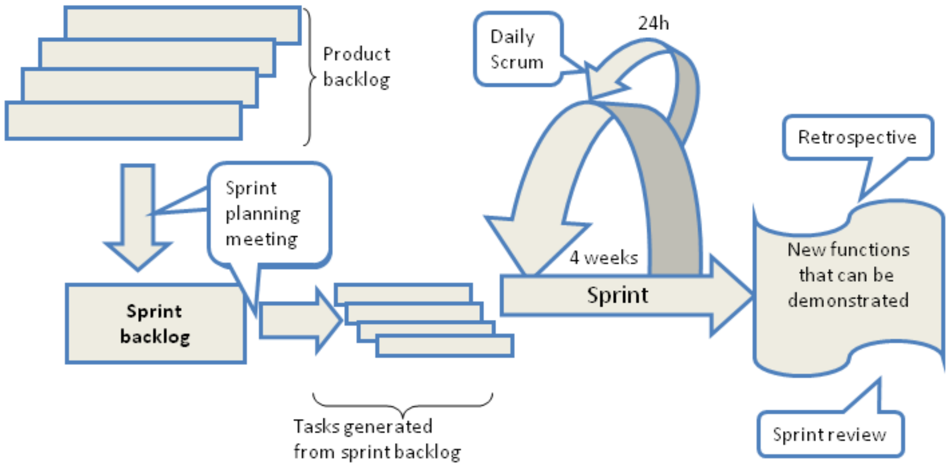

Fig. 1. Scrum process bases on sprints which have length of 2 to 4 weeks [2]

The team selects a subset of PBIs to the sprint and commits to finish those during the sprint. Sprint backlog is formed from those selected PBIs. Sprint backlog contains tasks that are required to complete the selected PBIs. Sprint backlog is owned by the team and no one else can make changes to it. Every day during the sprint, a brief (15 minutes or so) meeting called daily scrum is conducted. There each team member answers three questions: "What have I done since last daily scrum?", "What will I do until the next daily scrum" and "What kind of problems have I encountered?".

2.1 Scrum Roles

There are three different kind of roles in Scrum: *Development team*, *product owner* and *Scrum master*. Development team typically has from five to nine members. The team does not have traditional software engineering roles such as programmer, designer, tester or architect. The team is self-organizing unit and its members choose by themselves the tasks that they will do. Team selects items from the top of ordered product backlog which they will do during the next sprint. Team collectively commits to complete the tasks which they have selected. At the end of a sprint, the team should be ready with a potentially shippable product or increment to the product. It is also responsibility of the team to create work estimates for the tasks. Each task should be split to so small unit that it will take from two hours to two days to complete the task.

Product owners job is to optimize return on investment (ROI) for the enterprise. ROI comes from progress towards a vision of products for the enterprise. ROI may mean anything: customer satisfaction, monetary interests, environment values, etc. Product owner is the point of contact from the team to the customer, however product owner should never be representative of a customer. Product owner presents customers interests to the degree that is in line with the interests of the company. The most important task of a product owner is to manage the product backlog (PB). PB should be prioritized and up-to-date. In addition, product backlog items (PBIs) should be specified in detail, so that items are enabling the team to implement them. Product owner also decides if the outcome of a sprint is ready for shipping.

The *Scrum master* is responsible for making sure the team is as productive as possible. The Scrum master does this by sustaining and developing the Scrum process, by removing impediments, by shielding the team from outside disturbances and so on. Scrum master should keep list of encountered impediments that he/she should work on. This list is only for the Scrum master. The Scrum master also organizes and runs meetings: sprint planning, sprint reviews and daily scrum meetings. The Scrum master should not give technical input to the team nor direct their decisions in any way.

2.2 Scrum meetings

In the following subsections, different kind of meetings that take place in the Scrum process are described.

Product planning meeting
Product planning meeting is a meeting where all the stakeholders come together with their input for the product backlog. This meeting typically takes place in between sprints, or at the end of iteration n-1 in prepraration for sprint n.

Sprint planning meeting
In *sprint planning meeting* team, Scrum master and product owner are present. In this meeting team selects PBI from the top of the product backlog that they will do in the sprint. Team knows its velocity in story points. Velocity means the amount of story points the team can do in a sprint. According to this number they can choose correct amount of PBIs to take into the sprint. Team then splits PBIs in to tasks which are from two hours to two days in size (not estimated in hours tough) and puts those tasks in to sprint backlog. Planning poker should be used to create the relative work estimates for PBIs. No hours for estimation should be used.

Sprint review meeting
Product owner, scrum team and the Scrum master should participate in *sprint review meeting* along with management, customers and developers from other projects. The purpose of this meeting is to assess the project against the sprint goal determined in the sprint planning meeting. Team should not spend more than one hour preparing for the meeting. If the clients (PO and customers) are not satisfied with the results, the project will continue. It will be off by one sprint and people involved will inspect and adapt in the next sprint: what to do to meet the expectations of PO and customers?

Typically after sprint review meeting team has its own *retrospective*. The purpose of retrospective is to identify the things that the team is doing well that they should continue doing; things they should start doing in order to improve; and things that are keeping them from performing at the best and that they should stop doing. Typically product owner does not participate in this meeting, but the Scrum master participates.

2.3 Burn-down charts

Scrum uses burn-down charts as a central project management tool. The idea is to account for the time spent in the future. Time already spent can not be recovered and there is no point to track that. Burn-down chart helps the team to train they estimation skills, it shows how precise the estimates are. Scrum master also uses burn-down charts to see that the team is on track. If the team is not on track, the Scrum master can take corrective action: support a change of assignments within the team, bring resources to bear, etc. The burn-down chart should be updated daily. Team members should update their estimates of time remaining per task. However, one should consider using TRACK DONE [3] pattern with burn-down charts.

2.4 Concept of "Done" and technical debt

Defining when the task is done is important in Scrum. Usually a task is done when there is no remaining work meaning usually that design, coding, refactoring, documentation and everything else that adds business value is done. In the end, it is the responsibility of the team to agree what "done" means. The reason for this concept is that the product owner should be able to ship the sprint result immediately after the sprint is over.

Sometimes if the team falls behind near the end of the sprint, a bad Scrum master may push the release through, insisting that the team delivers the functionality while leaving re-factoring, code inspections, and documentation without attention. As refactoring is not a scheduled activity and because deployed code is rarely revisited, refactoring, code inspections and such rarely makes it onto the product backlog in future sprints. Eventually this work-not-done will slow team down. This work-not-done is called technical debt and it may accumulate from sprint to sprint.

3 Scrum Survey Results

In the following subsections, the results of the survey is presented. The results are accompanied with discussion on how the things should be done according to Scrum. However, one should take into account that company might have already inspected and adapted and drifed further away from the Scrum framework described in the literature.

3.1 Survey Overview

In this subsection, the main results of Scrum survey are presented. The survey was carried out by interviewing software architects, product managers, project managers and software engineers in Finnish companies. Interview questions can be found from Appendix 1. Interview questions included questions from so called Nokia test [1] (also known as Scrumbut test).

Originally the idea of Nokia test was introduced in 2005, when Certified Scrum Trainer Bas Vodde was coaching teams at Nokia Networks in Finland and developed the first Nokia test focused on Agile practices. He had hundreds of teams and wanted a simple way to determine if each team was doing the basics [4].

Nokia test cannot be taken as a scientifc method for evaluating how agile organization is or how well they are implementing Scrum. Even so, it has some value as it gives a rough estimate how Scrum is implemented. The results are comparable to each other as the grading for each question is carried out by the same person and with the same criteria. However, the results are not repeatable by other persons.

Nokia test consists of nine questions that investigates different aspects of Scrum: sprint length, concept of done, enabling specifications, product owner, product backlog, work estimates, sprint burndown charts, team disruption and team self-organization. During the survey these questions also revealed best practices to the interviewers.

It should also be noticed that the results do not present the overall situation in a company. It just reflects the practices of the teams that the interviewed person is involved with. If a person(s) from another team(s) had been interviewed, the results could be radically different.

Summarized results of the Nokia test are illustrated in Fig. 2. In Fig. 2 the results are average values of all questions of each company. Darker color presents results of companies that are participating in Sulava project. Lighter color present other companies.

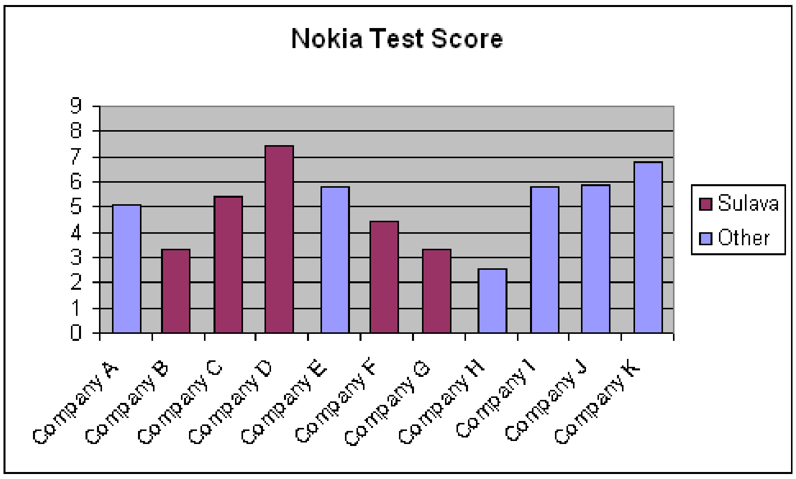

Fig. 2. Final scores from the Nokia test. Companies participating in Sulava project are drawn in red colour.

Table 1. Nokia test results in numbers

Company	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Average
A	10	7	10	0	6	5	0	3	5	5.1
B	10	4	4	2	4	1	2	2	1	3.3
C	10	5	8	1	4	5	7	1	8	5.4
D	10	7	7	6	6	9	8	5	9	7.4
E	10	3	8	3	7	8	7	1	5	5.7
F	10	1	5	4	4	5	3	1	7	4.4
G	2	5	6	0	4	7	3	3	0	3.3
H	10	5	5	0	0	1	2	0	0	2.5
I	10	1	4	5	3	8	6	10	5	5.7
J	10	7	4	4	6	8	0	6	8	5.8
K	10	9	7	6	4	8	4	6	7	6.7

According to Jeff Sutherland [1] typical Scrum Certification course average on Nokia test is 4 out of 10. At the end of a course, on average, participants think they can get their teams to score 6 out of 10 [1]. Table 1 presents the results to each question and overall scores as numerical data.

Scoring of each question was carried out by both interviewers (Veli-Pekka Eloranta and Jyri Vuorinen) separately. Afterwards scoring results were discussed and differences of two or more in scoring was discussed and resolved. In this way, misunderstandings in answers was avoided.

For other results of the test one might want to take a look at [5]. There are results from 778 (situation 19.11.2010) nokia tests. However, scoring from one to ten is not used there, but tracing back scores from the table 1 and by using the Appendix 2 one can have comparable results.

3.2 Sprint length

The first Nokia test question covers iteration length. The scores for this question are illustrated in Fig 3. Only one company was not able to score ten points from this question. Here ten points can be achieved by having fixed length iterations that last 4 weeks or less. Fixed length of a sprint is important as the release plan is created according to the sprint length. If the sprint length varies, no accurate release dates can be estimated basing on velocities of the teams and sprint length. Variable sprint length also is a signal that the team does not (or can not) really commit to the work they take into the sprint. The team does not need to deliver on a certain date, they just can make the sprint longer.

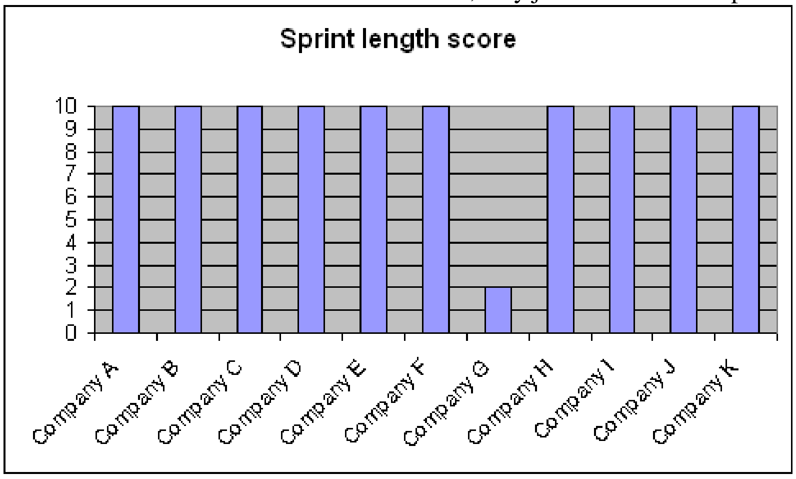

Fig. 3. Nokia test scores for question on iteration length.

Interviewed companies can be divided into two categories: to ones that have their own product (5 companies) and to those who do subcontracting and customer projects (6 companies). However, almost all interviewed companies reported that the most suitable sprint length is two to three weeks, while many companies prefered two weeks. Many of the companies started with four week iterations but found out that four weeks is too long. In many cases, priorities of PBIs changed drastically during four weeks. In addition, there were many disruptions, e.g. bug fixes, service requests, emerging requirements and such, that the team could not respond to. When the sprint length is shorter, these changes can be taken into the next sprint as the current sprint will end in two weeks at maximum. In this way, the customer or another department or another team requiring the changes can be kept satisfied with the response time.

Some companies reported that three week sprint is also too lengthy. However, none of the interviewed companies said that shorter than two week sprint was good. Some of the companies had tried sprints of one week or week and a half. The overhead caused by sprint planning and review was too big in such sprints and the teams switched back to two week sprints. In addition, some companies reported that the product owner or customer was too busy to attend to sprint review meetings that often.

Even though sprint length of two weeks was recognized to be good, there were a couple of problems with it. For example, if the team was working for multiple projects it caused situations where the team got so many change requests that they did not get anything done in a sprint. However, this problem is not related to sprint length but to product owner failure as all change requests should go through product owner to product backlog. Another reason why two week sprints were not used, was hour reporting. One company reported hours to the client monthly, so they have decided to do one month sprints even though they felt that the optimal length of a sprint is two weeks.

3.3 Testing in sprints

The second question of the Nokia test concerns the state of the software after each sprint. How it is tested within a sprint and is it deployable at the end of the sprint. Fig. 4 summarizes the results of this question. In this question there was a lot more divergence in the answers than in question one. However, all interviewed teams carried out unit testing within the sprint. This is more or less the result of automated tests. One company could have deployed the software after each sprint, but they did not dare to do so. This was due to two reasons: they did not trust the maturity of the software enough and the customer had refused to have new updates every two weeks. That's why the team created releases every now and then and did not deploy the software every two weeks. However, critical bug fixes were deployed immediately after a sprint when necessary. In this particular company also all documentation related to new features were ready at the end of the sprint.

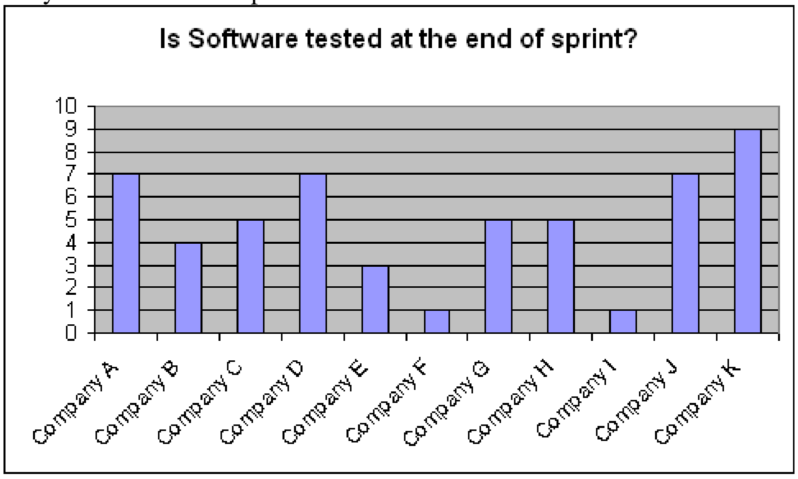

Fig. 4. Nokia test scores for question on what is the state of the software after each sprint.

In this question the scoring is a quite problematic as it depends on how one defines what the product is. If product is piece of code that some other team or company uses, how can you deploy that? In this case delivering the code to the team or to the client can be interpreted as deployment. However, this also makes a sprint review meeting hard as the code might be hard to demonstrate.

A typical problem in mobile working machine domain related to this question is that testing of the software takes very long. It might take over three months to test the software so that it is ready for deployment. Maybe in this case, concept of "done" should be redefined so that it is enough that the software is HIL tested (HW involved in the implemented feature) or tested in a simulator (in case of pure SW feature).

In many of the interviewed companies the Hudson [6] continuous integration tool was used to tackle this testing problem. Some companies also had added testers as Scrum team members in order to be sure that the software is tested and "done" at the end of the sprint. In many companies water fallish thinking still lived within sprints. During the sprint, the implementation was done first and then a week before the sprint review meeting the team started to test. Some companies even had separate sprints for implementation and testing. However, that is far from the Agile ideology. The features should be tested as you go - immediately after they are implemented. However, it might be useful to have some kind of feature freeze when there is say one day left in the sprint. In this way, there is time for the team to polish the code and fix possible bugs.

3.4 Enabling specifications

Third question of Nokia test is about enabling specifications. Accurate scoring of this question is quite problematic as in the interview no examples of specifications were seen. That created the problem how to distinct between poor and good user stories. Highest scores (above 7) were given when the company had just in time specifications that were ready just when they were needed. In addition, the specifications had links to other more detailed specifications and UI protos when needed. Figure 5 shows the results for this question.

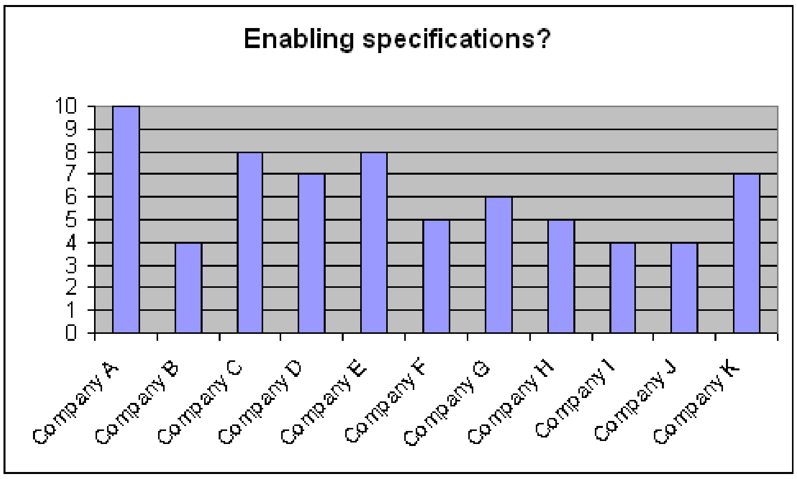

Fig. 5. Nokia test scores for question on how the requirements are documented.

Many interviewed companies used user stories linked with UI protos on small application projects. In larger projects or library projects feature-based product backlog was used. In many cases this product backlog consisted of features which had a hierarchy consisting of smaller features and functionalities underneath them. When these PBIs were taken into the sprint they were converted to tasks. In some cases also more detailed plans were attached to these features, these documents ranged from two to four pages. Especially when company had distributed development, more detailed specifications (separate documents) were required.

The problem in having just features in the product backlog is that the contents of the PBI may be unclear to the team. In some cases, if replacing old product with new version, team can look there what is meant by the feature. However, Product Owner should document PBIs in more detail. Having an architect in Product Owner team (see 5.4) may help in this. In the analysis phase the architect can create architecture and it typically helps to specify features in more detail. In this way, the specification will also be more accurate. However, one should be aware not to do too much specifications up front as it might be changed and therefore be wasted work. The same applies to large up front specification documents that some companies are still using.

Some companies used use cases instead of user stories as they saw that user stories did not capture some essential parts of the specifications. Use cases can be used in agile specification as well as user stories. In fact, in many cases there might be even a better option than light weight user stories. In addition a couple of companies used wiki for documenting specifications. In this way, any stakeholder can edit specifications when needed.

3.5 Product Owner

The most important and the most challenging role in Scrum is Product Owner. Nokia test question about the product owner is also a touchstone for how agile company really is. If a company has really adopted agile principles and Scrum process organization wide, there should be no problems playing the product owner role. However, the results in Fig. reffig:productowner show that organization are really struggling with Product Owner role.

Only a handful of companies had product owner which had product backlog ready when the team needed new PBIs. Many companies reported that the team does not know who the product owner is or they do not have this role at all. Some companies even reported that they consciously have not sent people to CSPO course, because they do not see any benefits that they can get out of it. However, it seems that the biggest problem with Scrum that companies are dealing with is related to Product Owner.

One notable observation was that companies which worked mainly with customer projects said that product owner does not work for them as they are doing customer projects. In addition, the same companies reported that the product owner role would work better if they were building their own products. However, companies that were working with their own products reported that the product owner role does not work well as they are building their own product and it would be easier if they worked with customer projects. In many cases, where the company did customer projects, the customer was communicating directly with the development team (Fig.7) or product owner role was assigned to a person from the customer. This led to a situation where the Scrum

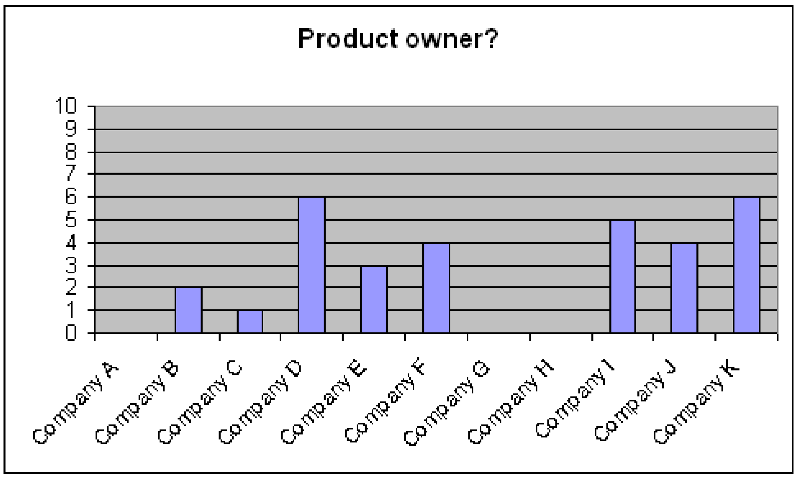

Fig. 6. Nokia test scores for question on product owner.

team is constantly disrupted by the customer. There is no buffer for emerging requirements and changes. Customer as Product owner forces the team to take changes within a sprint. Or if there is no Product owner at all, the Scrum team does not have required authority to say that the earliest moment that the request can be responded to is at the start of the next sprint.

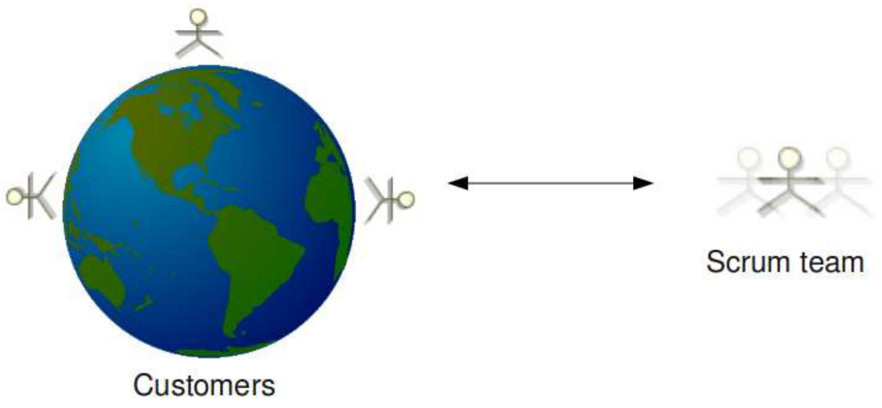

Fig. 7. Typical Product owner failure mode in companies working with customer projects.

On the other hand, in companies which work with their own products, it seems that typically the command chain from product owner and the development team to customers becomes very long (Fig. 8). In many cases there are existing roles such as product manager, project manager, sales personnel, etc. who do no fit so well with the Scrum organization. This results in the situation that Scrum is only implemented on the lowest levels of organization. The team is run as a Scrum development team and they may have Scrum master and product owner. However, Product owner does not have direct connection to the customer and to the end users. In many cases product owner also does not have enough authority to decide about the features of the product. The problem in this case is that information flow from customers (and from end users) to the development team becomes very slow or there is no communication at all. The development team is building the product almost blindfolded as feedback loops are very long and have a lot of indirection.

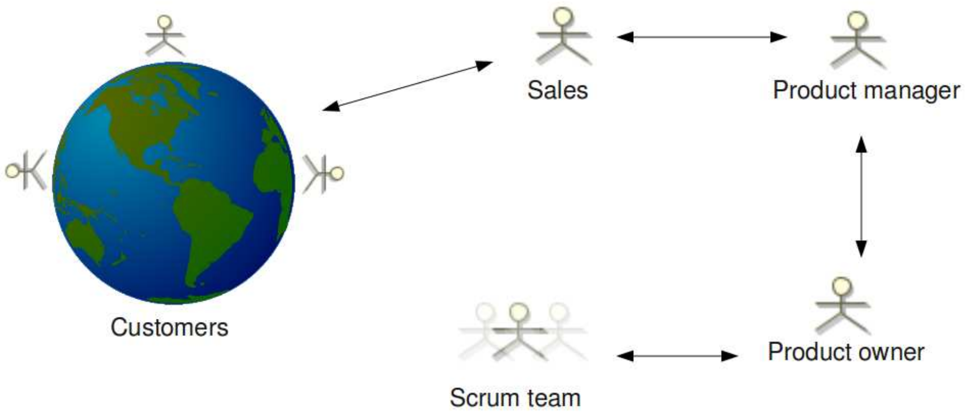

Fig. 8. Typical Product owner failure mode in companies that build their own products.

Fig. 9 shows how the development team and the customer should have only SINGLE MEDIATOR [7]. If the product is too big for one person to handle one could consider having PRODUCT OWNER TEAM [8]. The Product owner team has a chief product owner which is a facade for the product owner team, so it looks to the customers and to the development team like there is a single product owner.

Companies which work with customer projects should have their own product owner. There is a big risk that the customer representative does not understand Scrum and can not play along with the rest of the organization. If the customer insists to play the product owner role, companies could have proxy product owner of their own. Company's own product owner could work with product owner of the customer and act like the only product owner towards the development team. In this way, the Scrum team has local product owner that is available and can be asked for clarifications for PBIs and tasks. Additional wait states are eliminated and this kind of organization structure prevents the customer from disrupting the team.

Other problems with the product owner role that was found out in the interview was that product owner is too busy to write enabling specifications. However, this problem might be solved by using PRODUCT OWNER TEAM pattern [8]. Product owner should also use the development team to test if PBIs are splitted into small enough chunks. This can be carried out by having extra estimation sessions now and then and when too

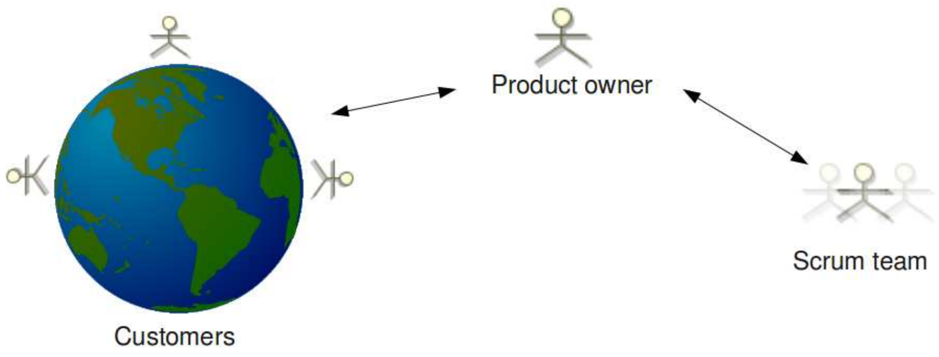

Fig. 9. Product owner role in Scrum.

big PBI is found, the Product Owner can also consult the team how to split the PBI into smaller pieces.

3.6 Product Backlog

The fifth question of the Nokia test discusses the product backlog. The results of this question are presented in Fig. 10. None of the interviewed teams based their roadmap and release dates on the team velocities and product backlog items. However, some teams faked the process by calculating internal release dates using velocities, but the actual release dates were still given from the management.

Some of the interviewed teams said that they have product backlog ready and items on it estimated for one or two sprints. However, in many cases there just was product backlog with user stories and interviewed people reported that the user stories are more or less ready to be taken into the sprint. So they may not satisfy "invest criteria" in many cases.

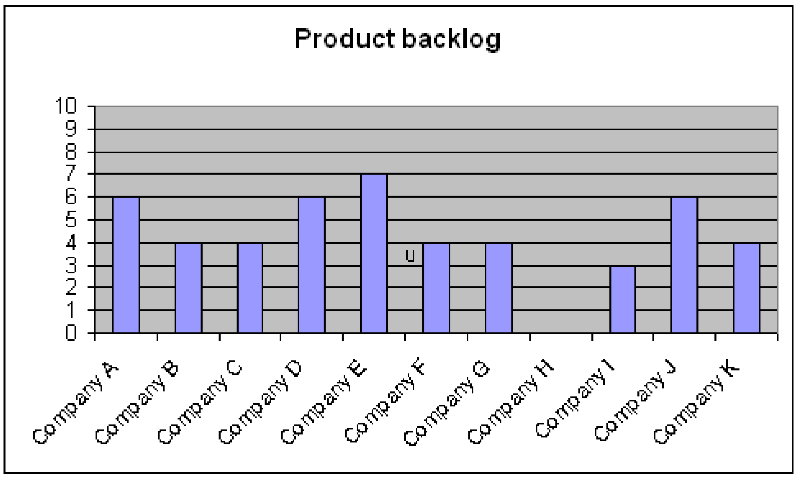

Fig. 10. Nokia test scores for question on product backlog.

In some cases, the company used timeboxed sprints and assigned PBIs to the sprints as they added new PBIs to the product backlog. The criteria to this was that the customer wants to have a feature ready at a certain date. This kind of approach is not however very agile as new requirements emerge, it may be hard to fit them into the sprints as sprints are already allocated for other PBIs.

In many cases the priority is determined by dependencies of PBIs. Product owner has to figure out the critical path and teams should start working on that. For example, there is no point implementing a search feature, if there is no way to input items that are searched for. In some cases also product owner set items that smelled like trouble higher in product backlog. Teams started working with potentially problematic items first. This hunch was based on experiences from previous projects. Calculating ROI is a hard job, especially for small PBI. Larger entities are easier to manage. Some product owners used theming and these themes were assigned a ROI value. Basically this means that features have ROI value and smaller items under the feature have the same ROI value. This makes valuing easier but there is no prioritization within a feature.

One problem with backlog approach that was found out during the interviews is that there is no room for innovation or work that the team feels is important, e.g architectural refactoring, etc. Some companies have solved this problem by having product owner who prioritizes some tasks important to the team higher, so that they eventually get done. Anpther practice that was found is to have TEAM SPRINT 5.5.

Almost all of the interviewed teams used Excel to admistrate product backlog. Teams that were doing distributed development had taken some other tool in use, basically because they were forced to do so due to the distribution.

3.7 Work estimates

The sixth question of the Nokia test is about work estimates. This question divided interviewed organizations into two categories: to those where the team produced work estimates and to those which used more traditional methods and managers produced work estimates. The results of this question can be seen in Fig. 11.

In most of the companies the Scrum team and developers are the ones who create estimates for the tasks and PBIs. However, there still are teams whose estimates are created by the project manager. The latter way of estimating may not lead to so accurate results as estimates made by using planning poker. One common pitfall is that the items are estimated just when they are taken into sprint. In this way, Product Owner can not form a release plan based on team velocities as there are no estimates of PBIs available. In addition, it might be noticed too late if some of the PBIs is too large for the sprint and should require further analysis.

One common problem with estimates in Scrum is that teams use hours for estimation instead of story points. This typically is result of hour-based billing. The customer is charged according to hours and estimates (or realized hours) are used for that. The basic problem is that the Scrum team is working for multiple projects (from multiple backlogs) and used hours can not be calculated using calendar time, velocities and story points. In some cases also story points do not feel right for the team, people are skeptical about the new concept. However, teams that have used story points and poker planning reported the experience solely as a positive one. Basicly this is a management problem

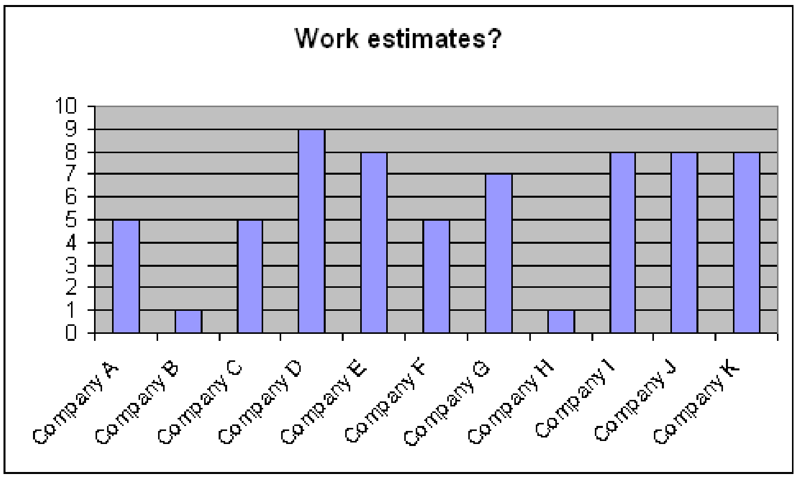

Fig. 11. Nokia test scores for question on work estimates.

- management are run by hours and story points do not fit in the organization. In some cases, teams had to do two separate estimates - one with story points and one with hours for billing. This kind of approach can be good during the transfer from hours to story points but one should be careful not to get stuck with it as two estimates creates two times as much work.

One good practice was that product owner arranges a separate estimation meeting during week one of the sprint (so called backlog grooming meeting). This two to three our meeting is for estimation only. The team estimates new and changed PBIs using planning poker. In this way product owner gets information if the PBIs are too large (takes more than 2 days). In addition, the product backlog gets estimated so it is ready for the sprint planning. When the whole product backlog is estimated, the product Owner can create the release plan according to the estimates and team velocities. Some interviewed companies even reported that if their Product backlog had not been estimated, due to special circumstances, their project faced some difficulties and unnecessary delays.

3.8 Sprint Burndown chart

Seventh question of the Nokia test addressed the usage of Burn-down charts. This question divided interviewed teams into three categories: to those who had embraced this new tool, to those who do not use charts and to those that used burn-down charts but used hours or did partial task burn downs. The results of this question are visualized in Fig. 12.

Burndown charts were quite widely adopted, only two companies did not use burndown charts. In one company there was also product burndown chart in use. Another one reported that they will probably take product burndown in use as PO has asked for it. Other companies reported that the burndown chart is most useful when used as a sprint burndown chart. Most of the teams used Excel to draw burndown charts. Excel was used as it made it easier to update the chart. Furthermore, in one case Excel was

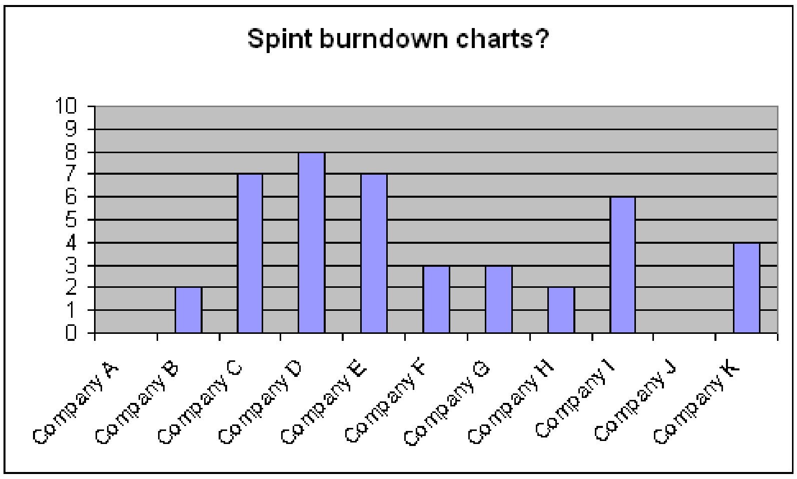

Fig. 12. Nokia test scores for question on burndown chart usage.

used as there was no suitable place for the paper chart in the office. Companies that did not use burndowns, reported that the benefits got out of the chart are not enough when compared to the effort required. Distributed development was the main reason to use other tools than Excel for burndowns.

Story points (that are used in burn down charts) are problematic according to many companies. If there is an existing strong organization culture using hours, it might be hard to transform that to using story points. However, using man-hours in burndown charts leads easily to partial burndowns or developers burn hours without really finising the story. This typically makes the burndown chart to look too optimistic and takes away the advantage of using it. Furthermore, hours tend to be developer specific. From some other developer, the task might require less or more hours. Story points support team thinking as the team can do a certain number of story points in a sprint. In addition, using hours may lead to situation where hours are always met, but this is achieved by taking technical debt that may lead to problems later on. In the worst case, this kind of behaviour can make the whole project delayed.

It seems that the main reason to go for using hours is to charge the customer on hours spent on their project. However, charging the customer can also be handled differently. As the developer works for a certain amount of hours in a month and the team can do a certain amount of story points in a sprint, the cost can be calculated from these facts. It can be calculated, how many hours are spent on a certain task proportionally. For example, if the task was five story points and a five person team did 300 story points in a two week sprint. This would mean that the cost of that user story is the cost of seven hours.

One small trick that came up in the interviews, is to get a new team into using story points is to say that a task that requires one day, is five story points. In this way, the team has common baseline for their estimates. After the first work estimation session, the team should forget that five story points were about a day of work and just estimate based on their experiences from past.

As well as story points, also the concept of velocity divided teams into two categories: some knew it, and some did not (or did not use at all). However, teams that used velocity reported some problems in using it. The biggest problem was that in many cases projects are quite small and short, and velocity takes some time to level up. In the beginning (with a new team), the velocity might go up and down before it stabilizes. In small projects, the velocity does not have time to stabilize. However, if the team members are the same from project to project and the domain is the same, old velocities can be used.

Another problem with velocity is that it requires that the team size stays stable. Team members might change, have vacations or sick leaves. This problem could be tackled by putting vacations in the sprint backlog and reserving some room for surprises such as sick leaves.

A good practice related to velocity that some companies used is that the velocity graph is also drawn. In this way fluctuations in velocity can be seen easily. If velocity is going down or leveling, it might be a sign that the team is in trouble and scrum master should take action.

3.9 Team disruption

The eight question is about team distruption: is the team disrupted during the sprint and if it is, by who. In almost all companies teams were disrupted during the sprint. Only in one company, team was not disrupted at all and all request for features bug fixes and such went through the Product Owner. The results of this question can be seen in Fig. 13.

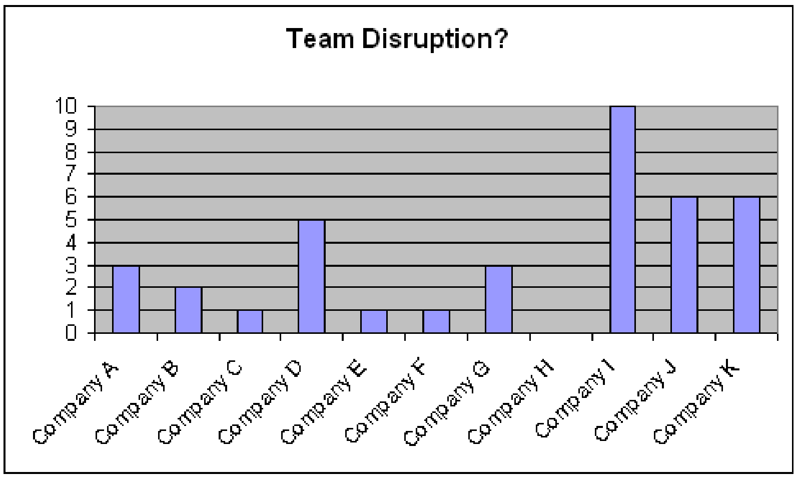

Fig. 13. Nokia test scores for question on is team disrupted during the sprint.

The problem here is that very rarely team members can be 100% dedicated to do only one thing. For example, members have worked in previous projects and they have to do maintenance tasks and bug fixes for the previous projects. These disrupts could go through Product Owner and from there to Product Backlog. However, it is hard to communicate to the customer that maintenance tasks and bug fixes take their time and the earliest point they can get attention is the next sprint. In some cases, the problem is that the customer does not understand agile methods and there is no SINGLE MEDIATOR [7] between the team and customer.

Team members may also have other commitments within the company. For example, one can be a member of the architecture team, process development group, etc. These may create random disruption to the team. In many case though, these could be added to the sprint backlog. In many companies, it had been learnt the hard way that things have to go through Product Owner to the backlog. However, there is still a lot of room for improvement.

Fortunately, there are a couple of things that came up in the interviews, that could be done in order to prevent the consequences of disruptions. Of course, in the long run impediment of disruption should be dealt with. However, in the mean time, the team can reserve 15 % empty space in the sprint to tackle with this problem. In this way, disruptions do not affect velocity and the sprint outcome. In addition, Scrum master could arrange QUESTION HOUR [9] to make it visible that there is certain time in the week when the team can be disrupted and they can fix bugs, etc. If it is Scrum Master or Project manager, etc. who disrupts the team, then the Scrum team must take the initiative to make this visible in the retrospective.

If the problem is that the maintenance tasks need to be done sooner, the team might also want to experiment with the sprint length. If the sprint length is four weeks, team could try to do two week sprints. In this way, emergent tasks get into the sprint with a shorter cycle.

3.10 Team self-organization

The last question is about team self-organization: does the team collectively commit to the sprint goals. Zero points in this question means that tasks are assigned to individuals during sprint planning by someone or team members do not have overlapping in their area of expertise. The goal with Scrum is to have emergent leadership within the team, however, everyone in the team is equal. Ten points in this question means that the team is in a hyperproductive state. That might be hard to detect during the interview, so that's why no team got ten points. Anyway, this does not mean that none of the interviewed teams is in a hyperproductive state. It's merely a limitation of the used survey technique. Results of this question are shown in Fig. 14.

This question revealed that in a couple of companies, even if some team uses Scrum, the tasks are given from project managers to the individuals. So there is no self-organization within the team. On the other hand, some of the interviewed teams were self-organizing. The benefits of this approach was that developers were not profiled as experts on a certain technology. This kept developers happy as tasks differed a lot. For example, they got a chance to work on lower level embedded stuff and sometimes on user intefaces. This approach kept the jobs interesting and meaningful and also developers wanted to stay in the same company as tasks were always changing.

One team had a chance to take this to the extreme. They did not have to do Scrum if they did not like it. They could use other methods as well, e.g Scrumban, Kanban, etc.

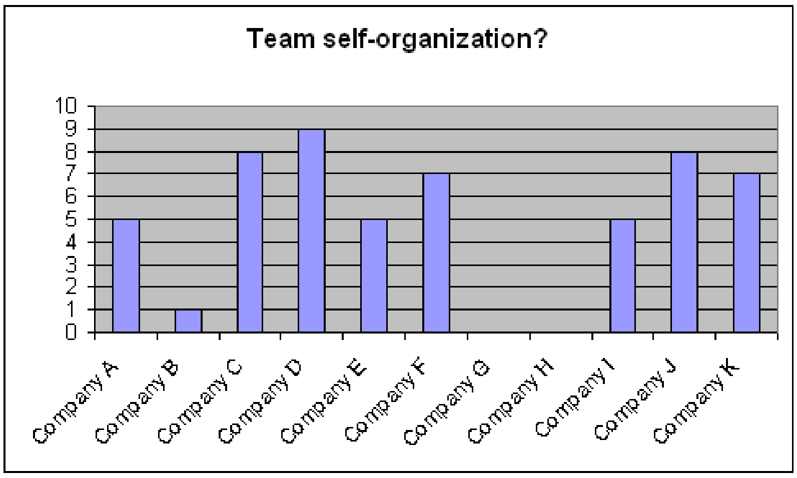

Fig. 14. Nokia test scores for question on is the scrum team self-organizing.

This kind of approach also provides the organization a chance to learn new methods. Teams who had changed from the approach where the project manager names the tasks to developers to the self-organization, reported that the change was only positive. In addition, when everyone is able to carry out any tasks, it makes work load balancing easier. Balacing can be carried out easily, even between the development teams. One team that is idle can help another, if they have enough expertise.

According to interviews, one problem was that expertise tends to profile on certain persons. For example, in one team, one developer was more experienced on the domain, so all architecture related tasks were always assigned to him. If this is the case, it might be a good idea to work on that task together as a team, so the expertise would be shared to other members as well. In one company, the architect role in the team was assigned by managers. This might damage the self-organization ability of the team. It would be better to have emergent leadership within the team.

4 Architecture and Scrum

4.1 Architecture work in Scrum

In software engineering, agile approaches are often contrasted with plan-driven approaches. This is sometimes misinterpreted to imply that planning is not required in agile software development. Obviously, careful planning is required in most industrial software development projects involving tens, hundreds, or even thousands of persons. However, instead of a rigid, detailed pre-existing work plan enforced during the entire process, in agile development there is typically only a fairly loose up- front project plan, and the plan is allowed to evolve and become more precise during the process. Software architecture is traditionally defined as the high- level organization of a system (e.g. [10], [11]). However, from the viewpoint of a soft- ware development project, software architecture is a plan of a new software system, in the same sense as a blueprint is a plan of a house. This creates certain tension between

Table 2. Number of teams using different approaches to do architecture work in Scrum

	Sprint 0	Architecture in sprints	Separated process
Number of Teams	7	5	2

software architecture and a non-plan- driven agile philosophy. Indeed, the role of software architecture design was questioned by the early advocates of agile approaches, claiming that software architecture emerges without up-front design efforts. Later, the role of software architecture in agile projects has been studied [12], [13], [14], but there is still considerable variation among practitioners concerning the ways architecture is involved in agile development. This is particularly visible in Scrum, which itself does not give any advice specifically regarding software related activities, like software architecture work. Lack of up-front planning is one of the main concerns about adopting agile and barriers for further adoption [15].

As stated in [12] architecture and Scrum does not go very well together. The problem is that there is no natural place for architecture work in Scrum. There is no sprint 0 in Scrum as every sprint must produce potentially shippable increment of the system [16]. One can not ship the architecture to the customer as it does not provide any value. So, if one does Scrum by the book, up-front architecture desing is not an option. What one could do is, do the architecture in the analysis phase before starting the sprints. However, analysis phase is mainly intended to be requirements gathering phase which is carried out by Product Owner. However, in many cases the Product Owner is a business person, who probably does not have required skills to design the architecture.

In addition, in the domain of working machines, Scrum teams have a lot of dependencies to teams that are not agile, e.g. to hydraulics engineering teams and to mechanical design teams. Hardware more or less dictates the architecture of the control system software as well. One has to take into account all requirements set by hardware before starting to write the control system. It is one thing to do things well, but it is critical to do the right things. Therefore, when a large complex system is being built, it is necessary to gather all significant requirements.

The second part of our interview survey concerned how teams do the architecture work while doing Scrum. Three different strategies were discovered: Sprint 0, designing architecture in sprints just-in-time and separated architecture process. Table 2 shows the number of teams that were using each approach. Note that the number of interviewed teams was larger than interviewed companies.

Sprint 0

Many teams do up front design, especially in the domain of embedded control systems. Some call it sprint zero, some call it architecture phase or analysis phase. The length of this phase depends on the complexity of the system. Basically these could be separated to two different approaches by their length. In the case of very complex control systems, the phase might last for 6 months and in simple systems it might be less than a length of typical sprints, i.e. less than two weeks. Just after this phase the actual implementation starts in sprints.

Typically in sprint zero or analysis phase, the requirements are gathered from hardware, electrical engineering, hydraulics engineering teams. During interviews teams reported that if they did not do sprint 0 they would be in big trouble. For example, if there is non-functional requirement that the software must be platform independent, it would be very hard to refactor the system to support this, if the requirement is not known from the day 1. In addition, control systems are so large and complex that it might be hard for one developer to understand the whole. Therefore, the up front architecture is needed to give context to the development work.

Almost all of the teams that used sprint zero reported that during the development sprints the architecture will still change. The most critical parts are fixed in the start and the rest of the design evolves. Typically in sprint zero system wide features are prototyped and tried out. Once the design covers these most critical features, it is ready for development sprints.

Architecture in Sprints

Four teams reported that they do not have separate analysis or architecture phase. They just start doing sprints and developing the system. The architecture is built in this process. In some cases, the design was guided by a single (or couple of) person(s) in the team that had the most experience on the domain. Another team members then built features on top of the design. One team reported that the designer is an emergent role within the team and changes from sprint to sprint. The same team also stated that anyone can do architecturally significant decisions.

The common nominator in this approach was that the teams were experienced in the domain. One company reported that they have had bad experiences with this approach when the team was new to the domain. They did not know what to do and as there was no upfront design the result was a failure. The architecture just did not correspond to the requirements and it had to be refactored all the time. This delayed the project significantly. If the team is inexperienced or architecture is not clear, the only option is to postpone the start of the project. In this case, the situation returns pretty much to what was explained in the previous section (sprint zero).

This approach is also a natural choice for a company that gets initial architecture plans from their customer. They just started developing and completing the architecture as they go. However, rarely application development works that way. The project type, where architecture is given by the customer is typically API development project or such.

The goal with this approach is to produce the initial architecture during the first sprint. It has details in those parts of the architecture that deal with the shippable features of the sprint. The rest of the architecture is completed in other sprints accordingly.

Separated process

The third way to carry out architecture work in Scrum was to have it as a separate process. In this approach architecture is created by a separate team that may have members from development teams but is completely separated from the development team. Members of this architecture team might do it part time, meaning that they are working also in a development team. The architecture team has checkpoints or milestones when certain part of the architecture must be ready. They may even make two or three different versions of the same architecture, so that there are options from where to choose in the checkpoint meeting.

In the checkpoint meeting, there are members of Scrum teams present. In addition, there are other stakeholders and at least one person who understands the business case. In minimum, there is the product owner, Scrum master, team lead and the architect present in the meeting. The customer might be also present in this meeting. In the checkpoint meeting the alternative architectures are presented and they are refined when necessary. Then plans are made, concerning how the architecture should be integrated to the system development (and when) or should it be implemented as its own branch. Checkpoint meetings are kept when necessary.

The thing here is that companies had completely separated architecture process from the Scrum process. In interviews, it was said that they don't want to mess the basic Scrum process in any way, and therefore kept the architecture process separated. This separate process might then produce tasks to the product backlog and connecting the separated architecture process to the development team. This approach might need technically oriented product owner to work.

Discussion on different approaches

There is not one secret how to do architecture in Scrum. It once again comes back to the basic Scrum principle: to inspect and adapt. If one is doing a relatively small project with teams having domain expertise, you probably won't need any upfront design of architecture. On the other hand, it might not be wise to start developing a very complex control system before doing rigorous analysis of requirements and without having upfront architecture design.

Many interviewed companies stated that Scrum has not brought any new problems to the architecture work. They said, it merely has surfaced existing problems and made them visible. Furthermore some companies reported that Scrum actually have helped them to create better architectures. In traditional waterfall model, one had to do big upfront design and just after that start writing the code. The Scrum has helped since the design does not have to be complete. Only large concepts should be fixed and details can be left open. Once the team has done a couple of sprints, the open issues are naturally solved as the experience grows.

One important thing that is easily forgotten is that architecture work should not be neglected after the initial design. Architecture deteriorates over time and it has to be refactored and updated. It is important while doing Scrum to be aware of this and add items to Product backlog that does the architectural polishing. Otherwise problems may emerge. Product owner should also know how to value these architectural PBIs so that they are implemented right on time. One option is to do so called team sprints (see Section 5.5). However, the interviewed teams said that Scrum has not increased the amount of architectural erosion.

However, there is one agile secret that can be applied also to architecture work: lean secret [17]. Lean secret is "Everybody, all together, from early on.". Agile often fails to value upfront design, however it is significant part of the development. The problem that has to be solved case by case, still is that how much upfront design is enough?

How to involve everybody, if the system is large, is another question to be answered. Probably all relevant stakeholders could be brought together when the architecture is designed.

4.2 Architecture documentation

Using Scrum has mostly affected specifications. User stories and use cases are used to describe the features of the system. Only the most complex parts of the system may have their own specification document. User stories might be accompanied with UI screens or API descriptions whenever it is suitable. However, Scrum has not affected architecture documentation so much. It just had made the role of architecture document more important, as according to interviews, the main specification document that is used. In many cases, there are several architecture documents as the systems are large and complex. There might be an architecture documents dedicated to single components of the system. More fine grained documentation is not typically used. Safety critical components or parts of the system might still have very detailed design documents as legislation requires them to be written.

The size of the architecture document varies. Typically it is 5 to 15 pages describing the high level architectural decisions. However, there are also companies that use larger architecture documents (as their systems might be more complex). In this case, the typical size of the architectural document is from 50 to 100 pages. According to interviews, larger documents are created if the team implementing the architecture is outsourced or inexperienced in the domain. All teams reported that Scrum has not affected the amount of documentation created during projects. It was widely recognized that it is typical misunderstanding of agile that one does not do any documentation. One company that are building their own product, lets developers decide how much documentation they want to produce. This approach had not caused them any troubles so far.

Many interviewed companies used Wiki such as Confluence to create the documents. It was preferred as it was self-organizing meaning that there are no ready-made templates that need to be filled. The problem with templates is that in many cases developers end up writing something that is not useful from the system under development point of view. In other words, companies that did not use templates felt that templates guide too much the way documentation should be done. Second benefit of Wiki according to interviews was that the newest version of documentation could easily be accessed and it can be written collaborately.

On the other hand, there are companies that use MS Word for documenting the architecture. In this case, typically document templates are used. Many companies that used Wiki for documentation reported that at the end of the project or when ending the analysis phase, they will make a Word document from Wiki pages. Some of the companies used also Doxygen or Javadoc whenever suitable to complement the documentation.

In general, some companies reported that it is impossible to produce very comprehensive documentation that would include requirements, functional specifications, technical specifications, testing plans, etc. If that kind of documentation was produced it would take all the time that is reserved for the implementation. So light-weight documentation is required. The most imporant thing in the documentation is to identify the potential readers of the document. Who is the target audience? The document should be then written for this audience. General architecture document is also good to have to give overview of the system. However, other stakeholder groups require different kind of documentation. For example, platform users do not need detailed description of the structure of the platform, but instructions on how the platform should be used and which are the things that the platform user should not do.

One problem that came up during the interview is that even though companies are using Scrum, the documents tend to be outdated. Architecture document is in many cases created and updated outside Scrum cycles and as a result the document gets outdated. Especially it was mentioned that UML diagrams tend to get outdated as it is more laborous to update them. Some companies reported that requirements and user stories in backlog get outdated. The problem with user stories is that the system is so large and new things come up in sprints, so user stories are not updated. Many companies working on embedded control system domain also reported that gathering all significant requirements is really hard as there are a lot of dependencies to other teams. In typical case these teams (electrical engineering, hydraulics, etc) have not even heard about Scrum.

One solution to this outdating problem might be to use really light-weight documentation that one company said that they use. They did documentation to Powerpoint by taking pictures of white boards and writing short stories to support the pictures. In this kind of approach, the document has to be polished (and maybe written with Word) at some point.

4.3 Architecture tools

Interviewed teams did not have any special tools for architecture work in use. Almost all teams used Microsoft Visio and Word to create architecture documents and designs. Some teams also used Wikis for documentation. UML tools were used as well such as Rhapsody UML or Dia. Powerpoint was also used to document the architecture.

4.4 Architect role

There is no role for an architect in Scrum, but many of the interviewed companies still had that role. In large projects there typically was an architecture team where architects designed the system before implementation in Sprints. Typically also, there was a system architect that was responsible for the overall architecture, hardware and software. However, this system architect was always a separate person from the architecture team and from the Scrum team.

When the designed system was large, the architect did not have time to code. However, on smaller projects, the architect was typically one (or two) person(s) from the team and participated also in the implementation. In these cases, the main designer was typically the person that was called as an architect. Other approaches existed as well. If there was a separate architecture team that gathered once in a while during sprints, the architect typically was a part-time role. Furthermore, this person listened to the team when selecting the technology that will be used.

A couple of teams also reported that the team is also allowed to make architectural decisions and these are then checked and approved by the main designer or architect. In one team, the case was that the Product Owner was the architect. Even though some teams were allowed to make architectural decisions, typically the design decisions tended to personate to one person. Everybody knew who was the main designer even though it was not officially communicated.

The common factor was that the architect was the person with the most experience in the domain. The architect must have good understanding of the domain and the system designed. Basically, the architect should be able to do any other persons job in team. This kind of approach also supports the self-organization of the team. It was also repeated in the interviews that the product owner should have some understanding on the architecture and architecture work in general.

5 Scrum Best Practices

During the interviews companies also reported good Scrum practices that they have adopted or invented. Some of them are generally known and used widely, while some of them are previously undocumented. In this section practices, that were not presented in the previous section, are introduced.

5.1 Daily Scrum

Daily Scrum should take place just before lunch. This has many advantages. In a company where people use flexible working hours, everybody should have arrived until the lunch and nobody has not still left. Another benefit is that when the daily scrum is held just before the lunch, everybody is hungry and rushing for the lunch. This keeps the meeting short as it should be. Furthermore, if there are some matters that need attention, those can be discussed over lunch.

5.2 Burn-down charts

Usually, team members update the burn-down chart daily to reflect adjustments to the amount of remaining work. These estimates are made in the middle of development time and they will show the increases of work amount that arise from emergent requirements. However, given that one emergent requirement has been discovered in a task doesn't imply that no others remain. On the other hand the Product Owner is not centrally interested in partially completed work, only in items that are done and potentially shippable.

Therefore, the burndown chart should only be updated in two cases: reducing the amount of remaining known work if the task is done and increasing the amount of known work if the task grows due to emergent requirements or other insights gained during the Sprint. The burndown should not burn-down on partial tasks, meaning that it should not be updated if the task is not done, done done.

5.3 Estimating product backlog items

The product backlog items should be small enough in the sense of man-hours. The optimal size of items is smaller than two days and still preferring to have smaller items. If an item is two days, and the development team is new or inexperienced in the domain the item might take even 16 days. So the item might take even longer than one sprint. The smaller the items in the backlog are, the more accurate the estimates get. One team reported in interviews that they had big problems with velocity and estimates when starting Scrum. Size of the PBIs was then typically from 1 day to 5 days. Then they made a rule that the item must be 2 days in size maximum or otherwise it has to be divided into two items. After this change the estimates were accurate and they could trust their velocity better. The secret here was that when the items were small, the developers had to think architectural changes, testing, simulations, etc beforehand, not just when taking the item under work. In addition, the dependencies between tasks became clearer.

5.4 Product Owner team

The Product Owner is accountable for many functions, and therefore sometimes a single Product Owner can't handle it all. If this is the case, the solution is to create a Product Owner Team that together formulates the Product Backlog. The Product Owner team should have a Cheif Product Owner that has final authority over the sequencing of the product backlog. The Product Owner team might also be a good option for a company that is in transition from old school methods to Scrum. Many people may feel that they are losing power and does not find their place in Scrum as there are only so few roles. By putting former project managers, product managers, etc on Product Owner team, they will find their place in Scrum.

5.5 Team Sprint

During interview one team reported that there is no room in Scrum for creativity and writing software is quite creative activity. The problem is that if someone makes an innovation or gets an idea for new feature during the sprint, they can not work on that. The idea is given to the product owner who evaluates it and puts it on the product backlog. However, if the business value of the item is rather low, the item might never get to the sprint. This is quite frustrating on the developers point of view as (s)he would like to work on his innovation. The same thing applies also for architectural work emerges during the sprints. Something changes and it might require architectural refactoring. If the product owner does not understand the value of architectural refactoring, it never gets done.

One solution to this problem, that came up in the interviews, is to organize so called **Team Sprint** [18]. This means that every fifth or tenth or so, the sprint is Team Sprint where the team can choose whatever items to the sprint they want from the Product backlog. Not only those items that are in the top of the PB. Team Sprint does not have to be regular: if there are more important things that has to be taken care of, the team sprint can be postponed. However, the Product Owner should not postpone it forever. It should take place as soon as possible after the emergent issues are taken care of.

5.6 Question Hour

A typical problem in the organization taking Scrum in use is that the teams get disrupted. It might be a support team, marketing, the customer, etc who disrupts the team. But the problem is that, the disruption interrupts the flow and therefore takes a lot of time that is reserved for development. One approach to tackle this problem is to organize so called **Question Hour** [9] once a week. This is the time that the team can be disrupted and they have reserved it for answering questions, fixing bugs from old projects, etc. This question hour is communicated to all stakeholders and it is made clear that this is the only time when questions can be asked from the team. Otherwise the team must not be interferred with. Over time, disruptions tend to decrease in number and questions are addressed more to Scrum Master and Product Owner.

5.7 Distributed Scrum

The survey also asked if a company did distributed development and what kind of problems it raised. The answers showed that in distributed Scrum, many new problems arise, for example, cultural differences, differences in organization culture, time zones, and the fact that face-to-face communication can be hard or impossible. Therefore, much more communication is needed. Furthermore, specifications must be really good, user stories might not be enough. Burndown charts must be in some online tool, so everybody has access to it. Also daily scrum might need to be held in a communication tool instead of face to face conversation. However, all people who are in one location could gather and have face-to-face communication locally and have people from remote location to join them. According to interviews, giving up daily Scrum is not a good option even when doing distributed development.

6 Conclusions

The adoption of Scrum practices varies a lot between different companies. Some have adopted all practices very widely and got better benchmarks in so called Nokia test. Some are still on their way of adopting the practices or have just taken few tools from Scrum to their own processes. As a surprise, the adoption of Scrum practices is not dependent on the company size. There are large companies that are very agile as well as small companies.

The typical problem with Scrum in interviewed companies was the Product Owner role. In some companies, there were no Product Owners at all. In some, the person who was the Product Owner, was too busy to provide the Scrum team proper backlog and enabling specifications. Creating a product owner team might help to the latter problem.

The most widely adopted Scrum practice was to have the work divided into sprints. The length of the sprints varied from two to four weeks depending on the product the company was building. The most of the companies used two to four week sprints.

Typical problems with Scrum in the domain of embedded control systems relates to testing and requirements. Testing in sprints is troublesome as test hardware (the work machine) might be ready after 6 months. The software should be ready in the same time as the machine itself. So the problem is that there is no real hardware where to test the software. Simulations can be used, but they do not correspond the real hardware and therefore the software is not yet ready to be shipped. In addition, in many cases the testing might take longer than a sprint as there are many safety features that has to be tested when some other feature changes. Another problem is gathering of the requirements. It is hard as there are a lot of dependencies to other non-software teams. Their desing change overtime also and requirements easily get outdated.

Architecture work has not changed much when companies have adopted agile practices. Documents are still written and architectures are designed. However, Scrum does not provide any role for the architect. Therefore, one has to think what kind of approach suits the company and the project best. Should the architecture be designed up-front or should there be a separated architecture team and process? Or maybe architecture can be built in sprints by the team. In the latter case, the team should be experienced in the domain or the results can be disastrous as some teams reported in the interviews.

A typical architecture document is rather small, 5-15 pages. However, some companies still create larger documents up to 100 pages. The problem with the large documents in agile is that they easily get outdated. Even if the teams update the document, there might be changes coming outside the team (e.g. from hydraulics team) and the document gets outdated outside the sprints.

In general, there is still a lot to learn about Scrum for the companies. There are also many topics of further research: How should the documentation be done in an agile manner? When should the architecture be created and how much upfront design is needed in the domain of embedded control systems? Which is the most efficient way of carrying out the architecture work in Scrum? How should testing be done when there are safety critical components included in the system design?

References

1. Sutherland, J.: Scrumbut test aka the nokia test (2010) Website, referenced 19.11.2010. `http://jeffsutherland.com/scrumbutttest.pdf`.
2. Eloranta, V.P., Vuorinen, J., Tommi, M.: Scrum in real life: Survey on scrum practices in small and medium sized companies. In: Submitted to SAC2012 conference. (October 2011)
3. Coplien, J.O.: Track done - published patterns (2011) Website, referenced 10.6.2011. `https://sites.google.com/a/scrumplop.org/published-patterns/value-stream-pattern-language/product-backlog/track-done`.
4. Sutherland, J.: Nokia test: Where did it come from? (2010) Website, referenced 19.11.2010. `http://scrum.jeffsutherland.com/2008/08/nokia-test-where-did-it-come-from.html`.
5. Vernois, A.: The scrum but test (2010) Website, referenced 19.11.2010. `http://antoine.vernois.net/scrumbut/?page=graph2&lang=en`.
6. Kawaguchi, K.: Hudson ci (2010) Website, referenced 26.11.2010. `http://hudson-ci.org/`.
7. Eloranta, V.P.: Single mediator - published patterns (2011) Website, referenced 29.1.2011. `http://sites.google.com/a/scrumplop.org/published-patterns/product-organization-pattern-language/single-mediator`.
8. Coplien, J.: Product owner team - published patterns (2011) Website, referenced 10.6.2011. `http://sites.google.com/a/scrumplop.org/`

published-patterns/product-organization-pattern-language/product-owner-team.

9. Eloranta, V.P.: Question hour - published patterns (2011) Website, referenced 23.2.2011. https://sites.google.com/a/scrumplop.org/published-patterns/team-pattern-language/question-hour.
10. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd ed. Addison Wesley (2003)
11. : Systems and software engineering - architectural description, draft (2010) Website, referenced 30.8.2011. http://www.iso-architecture.org/ieee-1471/docs/ISO-IEC-IEEE-latest-draft-42010.pdf.
12. Abrahamsson, P., Ali Babar, M., Kruchten, P.: Agility and architecture: Can they coexist? IEEE Software **27**(2) (2010) 16–22
13. Kruchten, P.: Software architecture and agile software development: a clash of two cultures? In: Proceedings of ICSE 2010. (May 2010) 497–498
14. Nord, R.L., Tomayko, J.E.: Software architecture-centric methods and agile development. IEEE Software **23**(2) (2006) 47–53
15. VersionOne: 4th annual state of agile survey, 2009. (2009) Website, referenced 30.8.2011. http://www.versionone.com/pdf/2009_State_of_Agile_Development_Survey_Results.pdf.
16. Sutherland, J., Schwaber, K.: The scrum guide - the definitive guide to scrum: The rules of the game (2011) Website, referenced 5.9.2011.
17. Coplien, J.O., Bjornvig, G.: *Lean Architecture for Agile Software Development*. Wiley (2010)
18. Eloranta, V.P.: Team sprint - published patterns (2011) Website, referenced 10.6.2011. https://sites.google.com/a/scrumplop.org/published-patterns/team-pattern-language/team-sprint.

Appendix 1 - Interview questions

In this appendix the interview questions are listed. Of course, more focused not-planned questions were asked during the interview session. In this way, we could extract more specific information from the interviewed people.

1. Name and position of interviewed people
2. What is the size of the organization?
3. Which kind of agile methods you have used? Xp, Crystal, Scrum, Lean, etc
4. How long you have been using Agile methods?
5. What is the typical team size?
6. What kind of software development you are doing? (Web, database, distributed, embdedded, etc)
7. Has your organization sent people to Scrum trainings such as CSM, CSPO?
8. Have you studied Scrum by yourself from books and other publications?
9. Has your organization document Scrum or agile methods in any way?
10. What is the length of Sprint?
11. Does your sprints end on a planned day?
12. Is the software fully tested and ready to be deployed at the end of sprint?
13. What kind of specifications you have? (How PBIs are documented)
14. Does the team know who is their Product Owner?
15. Do you have Product backlog that is prioritized according to the business value of PBIs?
16. Who (and how) estimates Product backlog items?
17. Do you use burndown charts? If you do who updates the chart and what is the used unit?
18. Does the team know its velocity?
19. Is the team disrupted during the sprint? If yes, who is disrupting the team
20. Is there any overlap in the expertise of team members?
21. Is the team self-organizing?
22. Who says how much work is taken into the sprint?
23. How do you take care of architectural desing in Scrum projects?
24. Do you use so called Sprint zero for architecture work?
25. Do you have architect role? If yes, is it a part-time role?
26. Do you have architect team?
27. Which tools do you use to support architecture work?
28. Has there been more problems with architecture in Scrum projects than in traditional (waterfallish) projects?
29. How do you document architecture and design?
30. Do you use architecture evaluations such as ATAM?
31. Do you do distributed development? If yes, experiences from that? What is the influence on architecture?
32. Do you have subcontracting in your projects? Does it affect architecture work?

In addition, there might have been some additional question in every interview.

Appendix B - Nokia Test

In this appendix the scoring criteria of Nokia Test is presented

Question 1 - Iterations

- No iterations - 0
- Iterations > 6 weeks - 1
- Variable length < 6 weeks - 2
- Fixed iteration length 6 weeks - 3
- Fixed iteration length 5 weeks - 4
- Fixed iteration 4 weeks or less - 10

Question 2 - Testing

- No dedicated testers on team - 0
- Unit tested - 1
- Features tested - 5
- Features tested as soon as completed - 7
- Software passes acceptance testing - 8
- Software is deployed - 10

Question 3 - Enabling Specifications

- No requirements - 0
- Big requirements documents - 1
- Poor user stories - 4
- Good requirements - 5
- Good user stories - 7
- Just enough, just in time specifications - 8
- Good user stories tied to specifications as needed - 10

Question 4 - Product owner

- No Product Owner - 0
- Product Owner who doesn't understand Scrum - 1
- Product Owner who disrupts team - 2
- Product Owner not involved with team - 2
- Product Owner has a clear product backlog estimated by team before Sprint Planning meeting - 5
- Product Owner with release roadmap with dates based on team velocity - 8
- Product Owner who motivates team - 10

Question 5 - Product backlog

- No Product Backlog - 0
- Multiple Product Backlogs - 1
- Single Product Backlog - 3

– Product Backlog has good user stories that satisfy the invest criteria - 5
– Two sprints of Product Backlog are in a ready state - 7
– Product roadmap is available and updated regularly based on team estimates of Product backlog - 10

Question 6 - Estimates

– Product backlog not estimated - 0
– Estimates not produced by team - 1
– Estimates not produced by planning poker - 5
– Estimates produced by planning poker by team - 8
– Estimate error < 10 % - 10

Question 7 - Sprint Burndown Chart

– No burndown chart- 0
– Burndown chart not updated by team - 1
– Burndown chart in hours/days not accounting for work in progress (partial tasks burn down) - 2
– Burndown chart only burns down when task is done (TrackDone pattern) - 4
– Burndown only burns down when story is done - 5
– Add 3 points if team knows velocity
– Add two point if Product Owner release plan is updated according to velocity

Question 8 - Team disruption

– Manager or project leader disrupts team- 0
– Product Owner disrupts team - 1
– Managers, project leaders or team leaders telling people what to do - 3
– Have project leader and Scrum roles - 5
– No one disrupting team, only Scrum roles - 10

Question 9 - Team

– Tasks assigned to individuals during Sprint Planning - 0
– Team members do not have any overlap in their area of expertise - 0
– No emergent leadership - one or more team members designated as a directive authority - 1
– Team does not have the necessary compentency - 2
– Team commits collectively to Sprint goal and backlog - 7
– Team members collectively fight impediments during the sprint - 9
– Team is in hyperproductive state - 10