

Anatomy of a Scrum Project

During the months of September and October 2003, I led a small but important (and problematic) Scrum project at a Swiss software house. The company specialized in point-of-sale software, and their product was used by (among others) the 2 major Swiss retail food store chains. The software, while quite stable, was based on a mid- to late-80's design and architecture, implemented in C and running on Unix (Solaris and HP/UX) with Oracle as a back-end.

Concurrent with the planned introduction in 2005 of a new bank/payment card system in Europe, the retailers wished for a general overhaul and extension of the system. Their demands and requirements (a modern, web-based interface on top of a J2EE architecture, using Tomcat and Struts, and running on Linux), caused the head of the company to become worried, and he asked me to come in and do a 2-day (p)review of the planned project, to ensure that the solution they planned to implement was the right one.

Getting the Job

I structured the 2-day review as a “pre-trospective” workshop, loosely following the techniques outlined in Norm Kerth's excellent *Project Retrospectives* [Kerth01]. After a preliminary meeting with management, I came up with the following list of topics for the meeting:

- Definition of the goals of the review
- Proposed functionality of the new product
- Procedure for evaluating the development platform and tools
- Decision criteria
- Risks and dangers of J2EE vs. MS.NET
- Definition of the development methodology under consideration of financial and personnel constraints
- Metrics for defining progress
- Evaluation of the development situation at the company
- Are corrections necessary?
- Discussion and Decisions

Although not listed as a topic, I took some time at the beginning to explain the rules of the game. I emphasized Kerth's “four freedoms”:

- You have the freedom to talk about the project the way you see it, rather than the way others want you to see it
- You have the freedom to ask about any puzzles
- You have the freedom to talk about whatever is coming up for you
- You have the freedom to say that you don't really feel you have one or more of the preceding three freedoms

and informed the group that I would be checking their “comfort level” on a regular basis.

During the section on defining a development methodology, I spent time explaining the ideas and principles behind agile process in general, and Scrum in particular. Both management and the development team were interested and fascinated with the fact that Scrum allowed them a maximum on flexibility while guaranteeing high quality. Later on, I’d find out what they meant by “flexibility”.

By lunch on the second day, we were finished discussing the topics on the agenda, and I asked for an hour’s time to prepare for my final presentation. In this final talk, I discussed positive and negative factors, risks and drag factors, and illustrated a way to start the project. Not surprisingly, Scrum fit in quite well here.

Positive factors were an open company culture, and the fact that the employees had been working there an exceptionally long time (the “youngest” developer had been there almost 3 years). Negative points were the fact that work situation was often quite chaotic and improvised, with 3 different “bosses” and multiple customers all demanding things from the developers.

A high-risk item was the fact that individual developers were often solely responsible for whole applications, thus lowering the “truck factor” to a dangerous level. As an additional risk metric, I used Joel Spolsky’s 12-point test [Spol00], with the following results:

- Do you use source control? - No
- Can you make a build in one step? - ~Yes
- Do you make daily builds? - No
- Do you have a bug database? - ~Yes
- Do you fix bugs before writing new code? - Yes
- Do you have an up-to-date schedule? - Yes, but not centralized and not updated
- Do you have a spec? - No
- Do programmers have quiet working conditions? - No
- Do you use the best tools money can buy? - Yes
- Do you have testers? - ~Yes
- Do new candidates write code during their interview? - Yes
- Do you do hallway usability testing? - No

Using the metrics described in the CSM training documents, I then listed the various drag factors influencing the developers. The data is in the table below.

Drag Factor calculation

Base factor	1.0
Complexity	
Technology	0.6
Requirements	0.2
People	0.2
Productivity	
Time together	0.0
Knowledge of technology	0.4
Knowledge of domain	0.0
Coordination & Integration	
Co-location	0.1
Distraction	0.2
Multiple team coordination	0.0
Coordination manager	0.3
Total	3.0

In addition to the metrics described in the Scrum literature, I added a factor of 0.3 resulting from lack of a “coordination manager” or, better said, ScrumMaster.

This technique provided me with a powerful and unprecedented way to tell management: “This is how long your project is really going to take, these are the reasons why, this is what you can do to cut down the time needed, and this is what it is going to cost you to do so”. One example: the team had a distraction factor of 0.2. This was because (for reasons we’ll get into later) customers had direct access to the developers. Management originally considered this “direct support” a good marketing argument for their software, but the developers suffered under it. I pointed out that in Csikszentmihalyi’s book *Flow* [Chiks90], the time necessary for a person to get into a state of flow and become completely concentrated on a task, is roughly 20 minutes. A person can be yanked out of this state in a few seconds, though, and normally requires ca. 10 minutes to get his thoughts straight and concentrate on a new, or on the same task. I then pointed out to management that, at the rate they calculated developer time, each such disturbance resulted in a base cost of ca. \$70 – before the developer even started working on the new problem.

As a result of this data, the company management asked me to come in on a 50% basis for 5 months to get the development process straightened out, to train the developers in new technologies, and to oversee a first proof-of-concept prototype.

Doing the Job

Based on information gathered in the pre-project review, management asked me to lead a 2-month initial project. The goals of this project would be to:

- get the developers acquainted with the new technology
- do a proof-of-concept prototype
- install and shakedown the Scrum process

Since the developers were essentially just “killing time” until the classic project management went through the motions of doing analysis and design, the cost of the project was the fixed costs of the developer’s salaries (4 developers plus 2 interns who we were supposed to put to good use), plus an old machine that was commandeered to be used as an application server, and a copy of a Linux distribution. One “former” developer, who still had development responsibilities in another project, but who had been pushed into customer contact, took over the role of product owner. Management was convinced that this team was adequate to deliver the solution.

This was much easier said than done. The developers, while good, had been working in C for many years, and were hopelessly generations behind the knowledge and experience necessary to produce the state-of-the-art prototype the customers expected. Remembering the old adage that it’s impossible to leap more than 2 technological generations at one time, I gave up my hopes of implementing a full XP process, and decided to get be satisfied with getting the technical basics in place. To this end, I dug out my old training materials on OO basics, Design Patterns, Responsibility-Driven Design and the like, and resolved on implementing (for starters) the minimal technical skills I deemed necessary:

- Version Control
- Unit Testing
- Release Management

While release management is part of the Scrum process, the other two items aren’t. My solution was simple: their introduction and training became items on the product backlog. (An aside: when implementing both Scrum and XP in an organization, I’ve often successfully used the trick of first implementing Scrum, and then planning the introduction of XP techniques as items on the product backlog). Considering the size and length of the project, and the necessity of having small feedback loops, I opted for a 2-week sprint length.

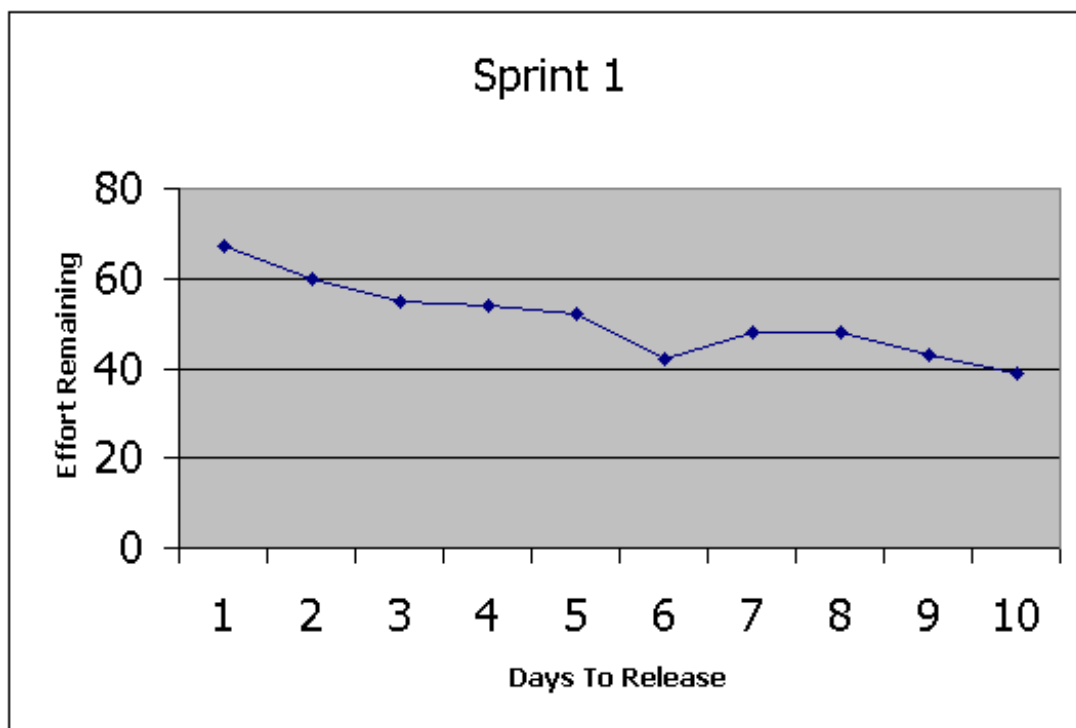
The technical level of the team was not the biggest problem, however. The issue which would eventually turn out to be the biggest problem confronting the team and the project – and actually the whole company – popped up its’ ugly head before the start of the first release planning meeting. Just before the meeting was supposed to start, one of the senior developers walked into the meeting room and informed me that he would not be attending the meeting, and that he wasn’t sure when he’d have any time.

As it turned out, this company had initially fought hard for market share, and had built up their reputation with the turn-around time for support. Unfortunately, this meant that

customers had the direct telephone numbers of the developers. Complicating this even more was the fact that, although the company and its' customer base was Swiss German speaking, the developers were all French, which meant that, even if they wanted to write documentation, they didn't have the language skills to do so.

In order to get the meeting going, we decided that the developer would stay for the beginning of the meeting, and then go off and solve the customer's problem. This turned out to be a minor issue. It seems that the customers got accustomed to the comfortable situation of not having to think for themselves, and were calling and disturbing developers for trivial issues. The developer was back within 2 hours, and happily worked on setting up the sprint backlog and kicking off the project.

As the developers signed up for and committed themselves to tasks, I realized quite quickly that they had no experience in agile task estimation techniques. Up to then, they had always been told what to do. Not knowing what their load factor was, they took on too much work. I quickly ran the numbers, saw that their overestimate of their own productivity would probably not stop the project from being finished on time, and decided to let them learn a lesson themselves, rather than stepping in and dividing their work up for them. The result can be seen in the burndown graph for Sprint 1.

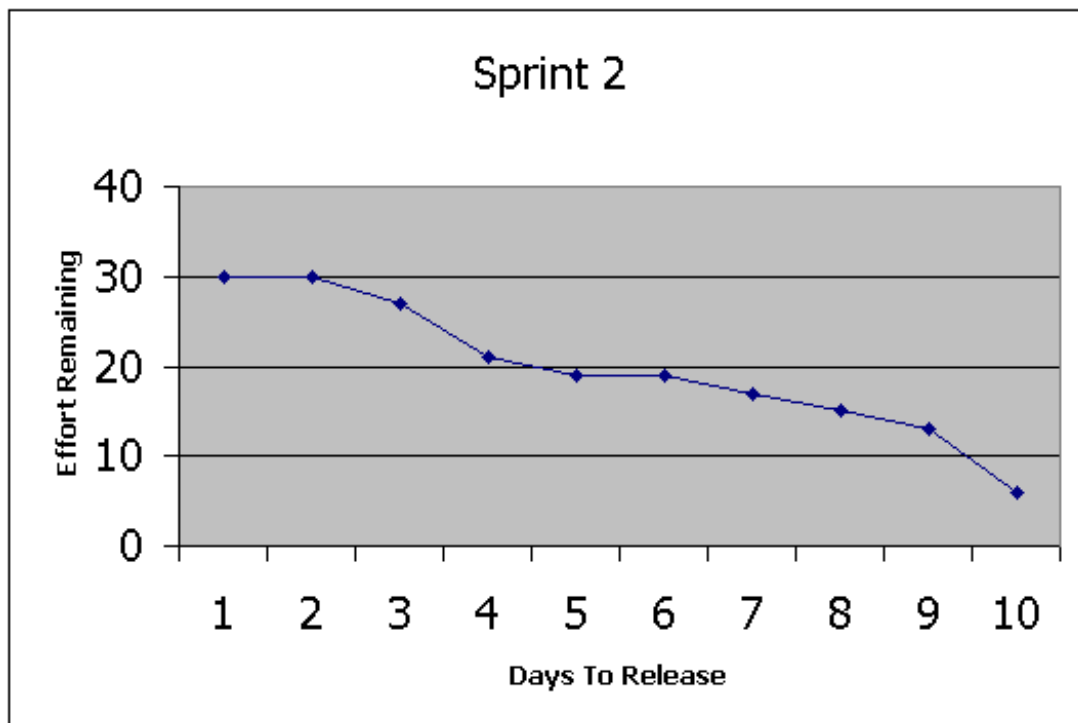


As can be easily seen, there was a great deal of work left at the end of the first sprint. (n.B. between days 5 and 6 was weekend.)

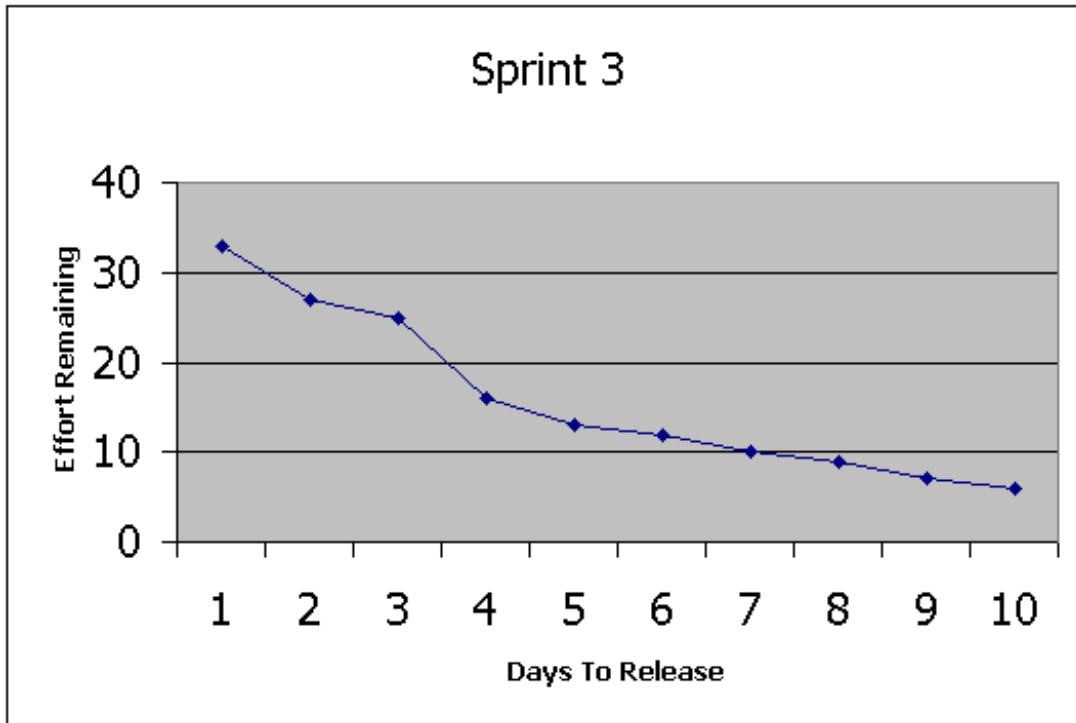
The problem with developer support responsibilities was a major issue. After 3 days of daily Scrums with different developers missing, I decided to change the work allocation procedure.

After each daily Scrum, the developers prioritised the remaining sprint backlog as “work on the table”. If a developer finished a support request, and had time to work on the project, he would grab the highest priority item from the table and work on that. This had the positive effect that the developers would often spend some time at the end of the day, or on the weekends, tackling some “work on the table”.

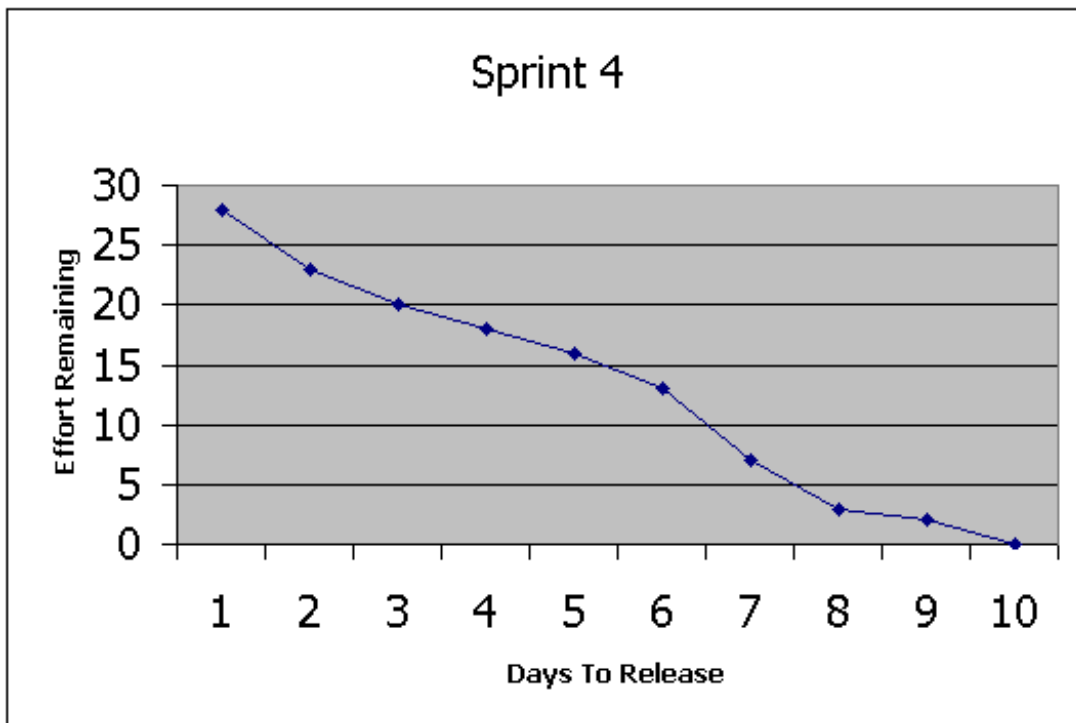
Every second day, I grabbed the product owner and showed him the backlog and burndown graph. He was amazed at how much information came out of the few minutes per day that the daily Scrum costed.



The second sprint was relatively normal. The first day was a holiday, and there were a number of small support requests towards the end of the second week. One thing noticeable is that the developers committed themselves to fewer tasks than in the first sprint, and that they worked overtime on the next-to-last day.

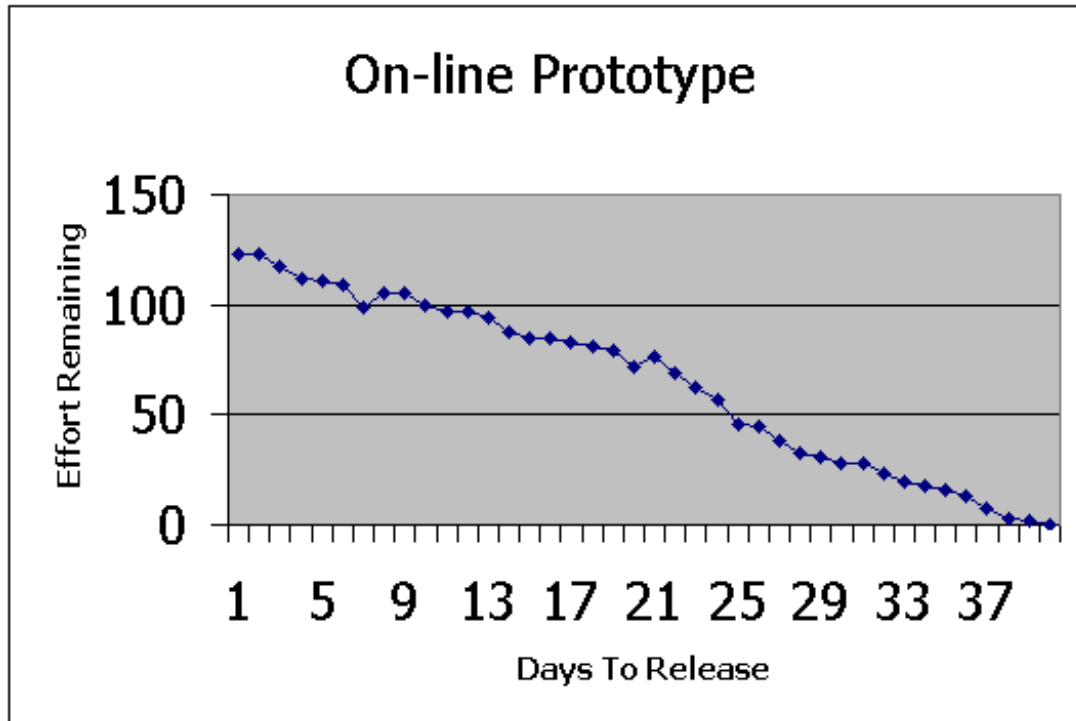


The major problem in the third sprint was that the CEO decided to get involved in the UI design. He insisted that the web interface to the application must have the same look-and-feel as the old curses-based interface. This cost the developers quite a bit of time, and slowed the development tempo drastically during the second week of the sprint.



The fourth sprint went along without any problems. The team saw the finishing line, and saw that they were going to be able to deliver the functionality. This caused a noticeable upturn on team morale and bonding.

Below is the burndown graph of the whole release.



Losing the Job

All good things must come to an end. Whether good or bad, the end that this project came to was unexpected.

At the end of the fourth and last sprint, the team proudly presented their prototype to management, to the internal product owner, and to representatives of the customers. With a clean, web-based interface, the prototype implemented enough of the basic functionality that the customers could get a feeling for how the finished product was (probably) going to look, and the team had enough experience implementing the technology to give reasonably accurate estimates to the tasks planned for the “real” project.

The day after the last meeting, I was called to the CEO’s office. He informed me that, although the prototype was finished on time, he was prematurely terminating my contract. Upon being asked why, he stated that he didn’t understand what I was doing, and that my work was not “transparent” enough.

As it turned out, he had seen me go into the meeting room with the developers every morning for the Daily Scrum, and had decided that he could do the same thing with the managers. Their meetings ended up taking 90 minutes per day, with no noticeable results, and he became suspicious of my work. I offered to come in and facilitate the management meetings, but was turned down with the response of “we can do that ourselves”.

My conclusion is that changes to an organization can be done most effectively (if not solely) at the level of Stacey’s “shadow organization” [Stacey96]. Working with developers, and being accepted by them for my development skills, was a major factor in effecting change. The failed attempt by management to implement a similar process was based in part on their lack on knowledge of the process, but also on the fact that they were operating strictly within the structure and hierarchy of the “official organization”.

Was the project a success? Yes. The team delivered a working system on-time and in-budget. The product owner and the end customers were happy with the result. Whether the positive experience using Scrum has initiated a change in the company is a question which I unfortunately cannot answer, but where I must presume that such an experience can not occur without leading to change – somehow, sooner or later.

Lessons Learned

Don’t let people get coffee etc. before the daily Scrum. This leads to delays in starting the Scrum. Also, with a good, compact daily Scrum, the need for conversation (i.e. a coffee break) will inevitably occur afterwards.

As a ScrumMaster, learn to sit on your hands – and on your lips. The team learns more if they discover solutions and solve problems themselves.

Don’t follow a process strictly simply because it’s described in a book. The basis of agility is inspection and adaptation. Try the book solution as a starting point if you don’t have enough experience, but don’t just do something because someone says you should.

References

- [Chiks90] Csikszentmihalyi, Mihaly, *Flow – the Psychology of Optimal Experience*, New York: HarperCollins, 1990.
- [Kerth01] Kerth, Norman L., *Project Retrospectives*, New York: Dorset House 2001
- [Spol00] Spolsky, Joel, *The Joel Test: 12 Steps to Better Code*, online at: <http://www.joelonsoftware.com/articles/fog0000000043.html>
- [Schwa02] Schwaber, Ken, and Mike Beedle, *Agile Software Development with Scrum*, Upper Saddle River: Prentice Hall, 2002.
- [Stacey96] Stacey, Ralph, *Complexity and Creativity in Organisations*, San Francisco: Berrett-Koehler, 1996.

Assignment

1. Describe one project on which you have used Scrum over the last twelve months. Describe:

- Purpose - what business goal was the project intended to deliver?
- Length - what was the duration of the project?
- Cost - what were the budgeted and actual costs?
- Value - what were the projected benefits and actual (if measured) actual benefits?
- Size - how many people were on the project team(s), how were they organized into teams?
- Teams - were the teams cross-functional and self-organizing? Were the teams collocated in an open space? Were the teams physically separated within one location, or located in more than one physical location?
- Initiation - how was the project initiated? How was the team trained to use the Scrum process?
- Reporting - how did you report progress to management and the customers?
- Change - what difficulties were surfaced by Scrum that had to be resolved? How were these resolved?
- Management - what was the previous role of the ScrumMaster? Who took on the role of Product Owner? To what degree were they successful in fulfilling their roles?
- Engineering - what software engineering practices or environment had to be changed?
- Stabilization - for how long did the software have to be stabilized before it could be released? How did you structure this stabilization process?
- Success - to what degree was the project successful? To what degree was the Scrum process instrumental in the success of the project?
- Scrum Process - to what degree was the Scrum process implemented "out of the box?" To what degree did you have to modify the Scrum process for this project? For each modification, how did you formulate the modification so that the basic inspect/adapt mechanisms continued to function? What parts of Scrum couldn't be implemented, or failed, and why?

2. How do you cause the accuracy of Product Backlog estimates to improve? To what degree does their accuracy matter?

3. How do you cause the accuracy of what a team commits to for a Sprint to what the team actually delivers?

4. What metrics do you use to track the development process? Which metrics have been changed, removed, or newly implemented as a result of using Scrum?

5. What type of training, resources, or tools would best help you successfully employ Scrum in the future?

6. (Optional) Scrum and Extreme Programming are sometimes used together. What must be considered when this is done?