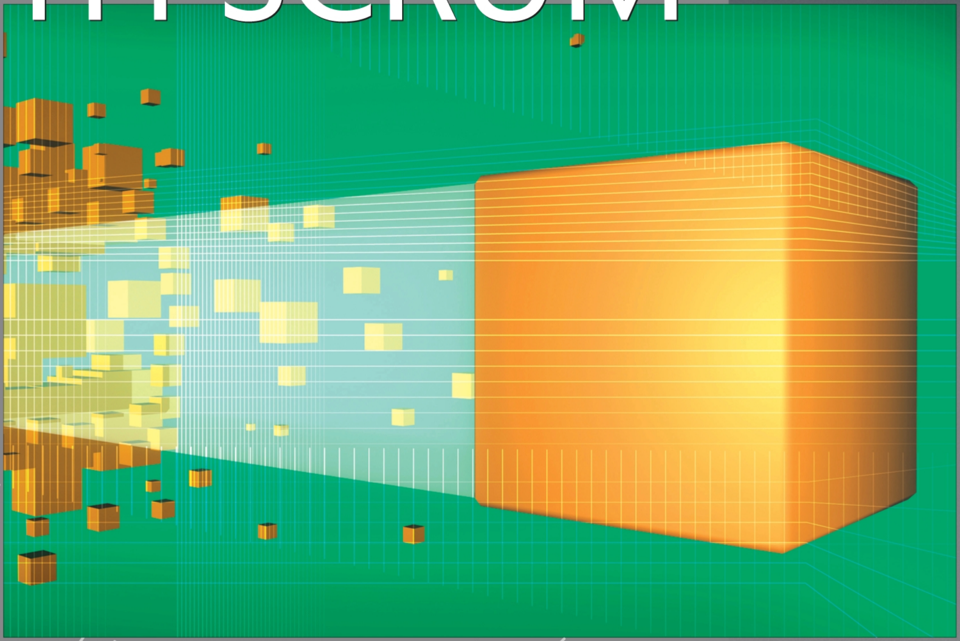
Microsoft

BEST PRACTICES

AGILE PROJECT MANAGEMENT WITH SCRUM



Ken Schwaber

Microsoft®

Agile Project Management with Scrum

Ken Schwaber

Russ thought about it for several seconds and realized that he didn't want to cancel the Sprint. Everyone would know that he was responsible for halting progress on the project for this minor opportunity. The Sprint planning meeting would make his act highly visible and provide his peers with an opportunity to ask why his pet project was more important than their needs. Russ thanked Terry but demurred, saying that he would meet with the Product Owner and get on the Product Backlog in the next Sprint planning session. Of course, he never did so.

Lessons Learned

Terry used the Scrum rules and practices to keep the project on track. Scrum offers many opportunities to make changes and to respond to new opportunities. Scrum also keeps everything highly visible. The Product Backlog and its prioritization are open to everyone so that they can discuss them and come to the best way to optimize ROI. The Daily Scrum keeps all team activities visible so that the ScrumMaster can enforce the rules and help the team stay on track. By keeping everything in full view, the type of backroom politicking and influence swapping normal in most organizations is minimized. These mechanisms are useful in bureaucratic organizations as a way to get particular things done. But when Scrum is already getting things done, these behind-the-scenes pressures are counterproductive.

Conclusions

At Trey Research and Litware, we saw that it's not always easy to understand the role of the ScrumMaster. At Contoso.com, we saw how a ScrumMaster can self-destruct. At MegaFund, we saw a ScrumMaster both fulfill his responsibilities and embed Scrum practices and rules in the organization. Something unique happened in each situation. The ScrumMaster was aware of Scrum's practices and rules and responded. Sometimes the response was good for the organization, and sometimes it wasn't good. In each instance, the ScrumMaster interpreted the job differently, and the results varied dramatically.

Over the last several years, I've wrestled with the question of how to make the difference between project manager and ScrumMaster, between coach and boss, more readily understood. How can I explain the shift in a way that is easy to absorb regardless of a person's background and inclination? When experienced Scrum practitioners are around to mentor a new ScrumMaster, the transition to Scrum is usually smooth. When I mentor new ScrumMasters, for example, I can help them understand many of the consequences of failure in part because I've failed so many times! I can also show them the difference between failure and success. We first fill the role of ScrumMaster ourselves, setting an example. Then we invite the new ScrumMaster to begin. We coach the new ScrumMaster after every meeting and throughout the day. We point out opportunities for the ScrumMaster to help the team. We point out ways that the ScrumMaster can tell when the team needs help. We also point out instances in which the ScrumMaster is controlling rather than guiding and explain what the consequences of such acts are likely to be.

The ScrumMaster is responsible for making sure that all the pieces of the Scrum process come together and work as a whole. The Product Owner must do his or her job. The Team must do its job. The chickens must be kept in line. The Product Owner and the Team must collaborate appropriately and use the Scrum meetings for inspection and adaptation.

The responsibilities of the ScrumMasters can be summarized as follows:

- Remove the barriers between development and the Product Owner so that the Product Owner directly drives development.
- Teach the Product Owner how to maximize ROI and meet his or her objectives through Scrum.
- Improve the lives of the development team by facilitating creativity and empowerment.
- Improve the productivity of the development team in any way possible.
- Improve the engineering practices and tools so that each increment of functionality is potentially shippable.
- Keep information about the team's progress up-to-date and visible to all parties.

When the ScrumMaster fulfills these responsibilities, the project usually stays on track. These responsibilities should be enough to keep the ScrumMaster busy; no ScrumMaster should have any time left over to act like a typical boss. Indeed, a ScrumMaster who acts like a program manager probably isn't fulfilling all of his or her duties as a ScrumMaster.

In my experience, some people intuitively understand the ScrumMaster role and take to it like a duck to water. Others struggle to understand Scrum and sometimes make harmful mistakes as they learn. However, even the successful ScrumMaster requires several Sprints to get going. When I am unclear about how to help a Scrum project, I've found it useful to keep the homily "the art of the possible" in mind. Focus on what can be done rather than be frustrated by what can't be done. This thought helps guide my actions at work on projects and in everyday life.

4

Bringing Order from Chaos

In software development organizations, chaos often results when a project's complexity is greater than its managers' ability to direct meaningful progress toward a goal. Progress might be made in fits and starts, but it is often indiscernible and unsatisfactory. Scrum cuts through this kind of complexity and wrests order from chaos. It does so by enabling a team to organize itself, which allows a particularly productive order to emerge. Let's visit several organizations to look at them before Scrum, to see how Scrum brought order to their projects, and to then generalize some practices from the experiences of these organizations. In these examples, we'll see the power of *time-boxing* to instill the art of the possible and avoid the pursuit of perfection, the practice of *incremental delivery* to improve engineering practices, and the practice of *empowerment* and *self-organization* to foster creativity and worker satisfaction.

The first organization we'll consider is Service1st, an independent software vendor of customer service applications that was introduced in Chapter 2. Service1st traditionally planned and managed its complex projects using extensive PERT charts. The results were less than stellar: the home stretch of every project was impressively chaotic and was invariably followed by an extended period of employee exhaustion and apathy. The second organization we'll look at is Tree Business Publishing. Tree's push to move its trade journals onto the Web coincided with several other big and messy initiatives, nearly paralyzing the development groups and causing extensive schedule slippage. The third organization is Lapsec, a research and development organization that builds proof of concept applications for the U.S. government. In the wake of September 11, Lapsec was called on to rapidly develop new information concerning potential terrorist activities. This project required melding a number of technologies and an untested capability called *data fusion*, an advanced agent-based form of data mining.

The Situation at Service1st

Service1st's development organization typically generated a new release of its software at least every year. The last two months of each development cycle were always a fire drill, and the result of each push for a new release was always an exhausted staff and buggy software. The company's managers had resolved to even out the intensity of the development effort over the six-month cycle, thereby relieving the development organization and improving the quality of each release.

Upon my arrival, the vice president of development took me on a tour of the engineering space. It was absolutely empty and completely still: perhaps one in four offices or cubicles was occupied. At first, I thought that it was so early in the morning that nobody had arrived yet. When I realized that it was already 9 o'clock, I considered that maybe a recent layoff had decimated the ranks. But no—Service1st was a growing and successful software company and hadn't had any layoffs since it was founded more than 20 years ago.

The vice president explained the situation to me: the company had just finished a six-month development cycle, the last release had gone out three weeks ago, and the staff was still totally exhausted. Developers had spent the last two months working evenings and weekends to complete the release in time. Not only was this bad for Service1st's employees, but it was also bad for its customers: because of the frantic pace of the last leg of each release's development, bugs often crept into the software and went unnoticed. The vice president said that he wanted to implement Scrum because he never wanted to put such demands on his staff again and because he wanted to improve the quality of Service1st's software.

What was the method behind this madness? How had the development staff gotten so overwhelmed? At the beginning of every development release, program managers coordinated with marketing to create detailed work plans for developing new functionality. The functionality list was derived from customer enhancement requests, high-priority bug fixes, performance enhancements, and competitive features. After a plan was established, modifications to the plan were handled through a formal change control process.

The work plans were represented in PERT and Gantt charts, with detailed resource leveling. The work was divided into numerous feature sets with high cohesion and low coupling, maximizing the proximity of work and reducing the dependencies. Anyone working on one feature set was unlikely to interact with someone working on another feature set. The only people for whom this isolated condition was not the case were those lucky souls assigned to work on multiple teams. Members of the development staff were given assignments and instructed to work on them until the release had been completed.

Complicating matters considerably, the development staff was assigned tasks by role; roles included analysis, design, coding, testing, and documentation. This method resulted in a waterfall way of thinking about work. One individual would analyze the requirement, another person would design the requirement, the next person would code the design, and then, finally, someone else would test the code. Rather than working together as a team, developers worked as though they were individuals at an assembly plant, passing the product along the line once they'd made their respective contributions. This method provided no opportunities for collaboration. Furthermore, the sequential nature of the work caused work to start and stop over and over again as people waited for one another to complete antecedent tasks.

Everyone in the development group had a lot to accomplish, so why wasn't the whole department hard at work at 9 A.M.? The vice president observed that the team usually didn't feel any pressure until three months before the release date and that members of the team started developing in earnest only during the last two months of the release cycle. Assignments at the task level, assignment of individuals to multiple teams, and particularly the waterfall approach all led everyone to feel isolated from the reality of the release during the first three or four months. During the last two months, the developers tried to make up for what they hadn't completed in the first four months.

The next release was due April 7 and was to be demonstrated at a user conference in March. Now was only the end of October. The staff was focusing on the upcoming holidays while recovering from the crunch of the last release cycle. Meanwhile, although it was only three weeks into the new release cycle, managers' anxiety levels were already high. Would the release be on time? Would the staff have to work like dogs again for the last two months? Nothing seemed to have changed, so everyone was expecting the worst.

Application of Scrum

Service1st's managers asked me to help them introduce Scrum as its development process. The company wanted all of the development organization transitioned to Scrum within two weeks. I started with the team that was working on a complicated piece of code to be incorporated into the next release: workflow capabilities.

Service1st had partnered with another software company and licensed its workflow products. During the development of this release, one team from each organization would work together to determine how the products would interact and figure out how to implement some workflow functionality. Members of the workflow team had been assigned tasks intended to further the design of four workflows. The folks in program management had selected these four workflows because together they represented a cohesive transaction.

The vice president thought that Scrum might be particularly effective with the workflow team because it was dealing with so many unknowns. He had scheduled a time for me to meet with the team so that I could get a feel for what they were doing and what progress they'd made thus far. I thought this would be a good opportunity to help the team understand a bit about Scrum; to that end, I conducted the meeting as a Daily Scrum.

The team members described their situation to me. Some of them told me that they were investigating how the two products could work together. The interaction between the products was intricate and complex. The team was struggling to determine how a single login could be used. Some were concerned that the security and authority schemes of the two products might be misaligned because each product had security mechanisms that were invoked based on user authority established at login. The team reported that although it had been working on this problem for three weeks, it could make only limited progress because the people who'd been assigned analysis work were still trying to determine how the products would be integrated in the first place. The people doing analysis were stuck, and so the other members of the team were sitting and twiddling their thumbs. They were spending their days redesigning their screens and databases to match the latest results from the people doing integration analysis.

I came away from this Daily Scrum meeting with the impression that this team didn't own its work. Someone had told the team what to do, and it was dutifully following instructions. I decided that a Sprint planning meeting would help the team focus its efforts on a small set of pressing problems and allow it to achieve some concrete results. I determined that I would ask the team to make the two products work together just enough to support one transaction that used workflow functionality. I asked the team to set aside the next day for a Sprint planning meeting, explaining that during this meeting we would figure out whether over the course of a 30-day Sprint we could build something that demonstrated integration of the two products.

We began work the next day at 9 A.M. by listing tasks the team was working on. We then wrote down the requirements for the four workflows from which tasks had been derived. After some prompting, the team members decided that the top priority was credit initiation workflow. They pointed out that work on the other workflows couldn't begin until the credit initiation workflow was complete. I questioned them about nonfunctional requirements that had to be addressed to support the workflow. We came up with the following list of requirements, both functional and nonfunctional:

1. Credit initiation workflow login
2. Credit initiation workflow startup