CHAPTER 2

Get Ready For Scrum!

Scrum is different. Work feels different. Management feels different. Under Scrum, work becomes straightforward, relevant, and productive. —

2.1 Scrum Is Different

I've spent a good part of my professional life building technology products and systems. I've had successes, and I've certainly had failures. I think I'm not alone when I say that most systems development projects are difficult. I suspect also that they are harder than they need to be. I remember a project when I worked with a plant manager at a pharmaceutical company. Together, he and I implemented a complicated material requirements planning system. As we were about to successfully complete the project, I congratulated him and told him that he could make a lot of money helping other companies implement similar systems. He looked at me aghast, and said, "I'll never go through something this gruesome again. I can't wait to go back to just managing the business!" His observation was one of many that led me to think that something was wrong; there must be a more straightforward way to build and implement systems.

Every project is different. The technology, the requirements, and the people are different every time. I've studied a variety of approaches to project management in an effort to make my life easier and the teams more productive despite their differences. I've tried new development environments, modeling tools, technologies, methodologies, people approaches, everything and anything to improve the process of building a system. I've found some things that improved my life, like always using the best engineers, forming cross-functional teams, and facilitating design sessions around white boards. These tactics all help, but without Scrum these projects were all eventually overwhelmed by the complexity inherent in systems development projects.

I once placed my hopes on commercially available methodologies. They contain templates of work that have previously been used to build systems. They therefore contain tried and true processes that other professionals have successfully used. Companies that build software for a living usually sell methodologies. I always assumed that for this very reason, the methodologies must be really good.

Methodologies are like cookbooks: follow their recipes and a successful system will result. Some methodologies are modest in scope and depth, while others contain literally thousands of pieces of work, or tasks, tied together into templates. Each template is appropriate for a specific type of development project.

Over the years I used commercial methodologies, they added definition to my projects. I knew what to do, when to do it, and I could assign people to the work. I felt like I was more in control and each project had a lot to show for it. Unfortunately, my success rate did not increase. One company that I worked at cancelled a major project after two years. I toured the project space not long after its cancellation and found a ghost town. There were hundreds of cubicles full of workstations and books of standards, training materials, requirements manuals, and design documents. Unfortunately, this project hadn't been successful. The project never even reached the software construction phase of the project, so no functionality was ever delivered.

As I mentioned earlier, I ran a software company in the early 1990's that developed and licensed a process management product called MATE. Our largest customers were Coopers & Lybrand and IBM, and they wanted us to employ their methodologies to build MATE. I attempted it and was thoroughly displeased with the results. At the time, my company's requirements were always changing and we were working with new technologies. It looked like the methodologies should help, but instead they just got in our way, decreased our flexibility, and generally slowed us down.

I wanted to understand the reason why my customers' methodologies didn't work for my company, so I brought several systems development methodologies to process theory experts at the DuPont Experimental Station in 1995. These experts, led by Babatunde "Tunde" Ogannaike, are the most highly respected theorists in industrial process control. They know process control inside and out. Some of them even taught the subject at major universities. They had all been brought in by DuPont to automate the entire product flow, from forecasts and orders to product delivery.

They inspected the systems development processes that I brought them. I have rarely provided a group with so much laughter. They were amazed and appalled that my industry, systems development, was trying to do its work using a completely inappropriate process control model. They said systems development had so much complexity and unpredictability that it had to be managed by a process control model they referred to as "empirical." They said this was nothing new, and all complex processes that weren't completely understood required the empirical model. They helped me go through a book that is the Bible of industrial process control theory, *Process Dynamics, Modeling and Control* [Tunde] to understand why I was off track.

In a nutshell, there are two major approaches to controlling any process. The "defined" process control model requires that every piece of work be completely understood. Given a well-defined set of inputs, the same outputs are generated every time. A defined process can be started and allowed to run until completion, with the same results every time. Tunde said the methodologies that I showed him attempted to use the defined model, but none of the processes or tasks were defined in enough detail to provide repeatability and predictability. Tunde said my business was an intellectually intensive business that required too much thinking and creativity to be a good candidate for the defined approach. He theorized that my industry's application of the defined methodologies must have resulted in a lot of surprises, loss of control, and incomplete or just wrong products. He was particularly amused that the tasks were linked together with dependencies, as though they could predictably start and finish just like a well defined industrial process.

Tunde told me the empirical model of process control, on the other hand, expects the unexpected. It provides and exercises control through frequent inspection and adaptation for processes that are imperfectly defined and generate unpredictable and unrepeatable outputs. He recommended I study this model and consider its application to the process of building systems.

During my visit to DuPont, I experienced a true epiphany. Suddenly, something in me clicked and I realized why everyone in my industry had such problems building systems. I realized why the industry was in such trouble and had such a poor reputation. We were wasting our time trying to control our work by thinking we had an assembly line when the only proper control was frequent and first-hand inspection, followed by immediate adjustments.

Based on this insight, I have since formulated with others the Scrum process for developing complex products, particularly software systems. Scrum is based on the empirical process control model. For those interested, more details on why Scrum works are presented in Chapter 5: *Why Scrum?* and Chapter 6: *Why Does Scrum Work?*

Scrum is a way of doing things that is completely different from what most people in the software and product development industry are used to. All of the assumptions, mechanisms, and ways of looking at things are so different that a new way of thinking evolves as you begin to use Scrum. Scrum feels and looks different because it is based on the empiricism. Less time is spent trying to plan and define tasks, and less time is spent creating and reading management reports. More time is spent with the project team understanding what is happening and empirically responding. Most people really understand Scrum only when they begin to use it. A light bulb goes off when they experience its simplicity and productivity. They realize how inappropriate more traditional models of development process are for our industry.

The following case study covers an implementation of Scrum and the application of empiricism. In it, I describe working closely with a team to build a product while using the Scrum process. In this example, I made decisions and encouraged the team to act differently than they were used to acting. I taught them by example to approach their work in an entirely different way. By the time we had completed the first Sprint, the team was already behaving differently. They had seen Scrum work, and now they were Scrum users. They had come to embody the values integral to Scrum, such as empiricism, self-organization, and action.

As you read the case study, think about what is missing from it. There is no formal project planning phase. There aren't any Pert charts. There are no roles and individual assignments. Notice how the team is able to get on with its work and build valuable product increments anyway. Notice the team self-organize from a dispirited group of individuals waiting for instructions into a team that takes the initiative and acts. By the end of the first Sprint, the team had adopted a completely new set of values and begun to act unlike any other team at the organization.

2.2 A Noisy Project

The project was to build a middleware business object server and its accompanying business objects. A large financial institution wanted to develop the product to connect its online transactions to its legacy databases. The institution needed to handle increasing transaction volumes, to standardize database access, and to carry out the implementation of new technologies such as telephone, wireless, and handheld input devices. This technology was all devastatingly complicated, including choices and learning curves for object technology, transaction management, hardware, operating systems, and development environments. To complicate matters, this was a technically sophisticated company, so proponents for various alternatives to each technology choice were numerous and vociferous. Furthermore, team members were working at multiple locations, and the team therefore needed to use a multi-site development environment technology. It had chosen to use an enterprise-wide code management software, but had not yet begun to do so.

The project was truly hellish. A development team had been chartered and charged. When I first began working with the team, it had been in existence for four months, but had not built any product. It was waiting for a budget. It was waiting for funding for new servers, for the last team members to be assigned, for the code management software to be licensed, and for someone who knew how to administer the code management software to be hired.

To begin implementing Scrum, I started holding Daily Scrum meetings. These meetings are supposed to be quick status updates. This was not the case at the first Daily Scrums. The first meeting took three hours, rather than the customary fifteen minutes. Everyone was completely dispirited and demoralized. Team members talked not about what they were doing, but about what was preventing them from doing anything. Many people complained that management didn't support the project, and everyone was upset the budget hadn't been formalized. Without a budget, the team couldn't order servers or license the code management software. For that matter, the team couldn't attract new team members, since it looked as though it was going nowhere fast. The team was without funding, without a sponsor, and without the tools that it needed.

2.3 Cut Through the Noise By Taking Action

One of the fundamental principles of Scrum is "the art of the possible." That is, Scrum instructs teams not to dwell on what can't be done, but to think about what can be done. Teams are put in a time box and told to create product. It is important to focus on what can be done and how the problem can be solved with the available resources. This team had a name, a scope, and definition, and it was staffed with some really solid engineers, all of whom had workstations and access to a lot of software. I asked the team what it could do with the available resources. I also asked the team whether it believed that the problem it was trying to solve was important to the organization.

The team confirmed that the problem was real and it was eager to tackle it. Some team members were aware that a customer service project was being held up by the very problem they were supposed to solve. The customer service project was supposed to implement access to the legacy databases, but was unable to proceed because this team had not yet built the middleware server that would handle legacy database access. Clearly, this team had been chartered because of a critical organizational need, and it had an important mission to accomplish. Until the team could get moving, other projects would continue to be held up.

The team quickly identified a core set of transactions that the customer service project needed it to enable. The team members felt that they had enough skills to build a middleware object server to implement these transactions, so long as someone from the customer service team worked with them as a domain expert. They felt they knew AIX, Tuxedo, and CORBA well enough to use that technology to implement the solution. They "borrowed" an RS6000 server from the server room to develop and prototype their work. The project manager, Herb, presented this plan of attack to his management. Since this effort required no additional funding and no administrative action, Herb was authorized to proceed. I got together with the team and devised a goal for the first Sprint. The Sprint goal was:

Sprint Goal: to provide a standardized middleware mechanism for the identified customer service transactions to access backend databases.

The team figured out the work they would have to do to meet the Sprint goal. The following tasks came up:

Map the transaction elements to backend database tables.

Write a business object in C++ to handle transactions via defined methods and interfaces.

Wrap the C++ in a CORBA wrapper.

Use Tuxedo for all queuing, messaging, and transaction management.

Measure the transaction performance to determine whether scalability requirements can be met.

2.4 Self-Organization

After identifying these objectives, the team began the Sprint. Since the team was using familiar technology, there were no major technological problems. However, two team members were at a remote site. Because the team didn't have an enterprise-wide code management system, it couldn't readily do multi-site code management. This problem was resolved by partitioning responsibilities between the two sites, and verbally coordinating whenever either site had to use code under the other's control.

The team met and decided who would do what work. When one team member wanted to work with the Tuxedo expert to learn the product, the team figured out how the rest of the team could pick up the slack. As the team started doing the work, it would meet frequently on its own to design the product and further identify and parse the work. The team did this on its own. It knew the Sprint Goal and its commitment. The team was figuring out how to live up to its commitment.

2.5 Respond Empirically

After ten days, the team started to feel like it was going to fail. The technology was up and working, it had figured out the CORBA wrapper, and it had accessed the appropriate databases. However, the team felt that it couldn't get the entire selected customer service transaction set mapped and linked to the database within the Sprint. The transaction data was too complicated and involved too many tables and indices for the mapping to be completed in thirty days.

The team had incorrectly anticipated the complexity and the scope of the work it had assigned to itself. But had it failed? Not in the eyes of Scrum. Working with a host of difficult technologies and unknown transactions, the team had built the development environment, put up a middleware server using Tuxedo, and had started implementing the customer service transactions. It was doing great. The team had done the best that it could rather than sitting around and doing nothing.

Again, I focused the team on the art of the possible. What could it do within the Sprint and still meet the goal? The goal wasn't to complete the entire transaction set, even though that was what the team had expected to be able to do. The goal was to prove the viability of a middleware object server providing database access to the customer service transaction set. No one even knew whether management would approve and fund this approach. The team quickly identified that they could address a reduced scope of transaction data elements involving fewer tables and indices, and then proceeded to automate this.

2.6 Daily Visibility Into the Project

On the fourteenth day of the Sprint I held our Daily Scrum. When it came to Tom's turn to report, he indicated that a Senior-Vice President, Lou, had instructed him to build something that was not within the scope of work for the Sprint. Consequently, he had been unable to do the work that the rest of the team had expected of him, though he would try to catch up. I immediately went to Lou's office and asked what was up. Lou had been offsite and had learned that a potential customer was interested in additional functionality. He had decided to help the team out by instructing one of its members to start developing that functionality.

Lou hadn't been at all of the Scrum training, so he didn't know that interrupting a Sprint is almost always more counterproductive than it is helpful. Lou didn't know that the team was protected during the Sprint from external chaos, complexity, and uncertainty. Lou said that if he saw a $100 bill on the ground on the way to the train, he would bend over and pick it up, and that he didn't see how this situation was any different. I told Lou that, in the greater scheme of things, his family would probably appreciate his getting home on time more than the $100. I explained to Lou the importance of not disrupting a Sprint, and he agreed to refrain from doing so in the future. By the end of the Sprint, the feature that Lou had wanted to be demonstrated was no longer on the radar of this potential customer anyway. Apparently, it had only been of interest to the customer as a conversation topic with Lou at the offsite.

2.7 Incremental Product Delivery

At the end-of-Sprint demonstration, the team really impressed management with its pragmatism and empiricism. With only the resources it had on hand, it had proven that its approach was technically feasible. In fact, it had put the technology to use for customer service functionality. Although a thorough requirements study might eventually have uncovered better technical approaches, the team had used available resources to solve the problem both for the customer service team and for the company as a whole. The team had been productive with what was on hand.

The team had run performance measures on its solution and proven that the approach could handle the expected transaction volumes. In an online session, it showed management part of the transaction going through the middleware to the databases, retrieving and displaying selected data, and doing so with performance and scalability that could be sustained.

The team presented an increment of product that was successful, could be discussed, and could be built upon. If the team had not gotten its act together, the organization as a whole would have been thirty days closer to a transaction volume meltdown. Instead, because of their empiricism, effort and initiative, the organization had something that worked and that could be modified and built upon. Incremental product delivery can be very powerful, providing an organization with real progress in a short period of time. Previously, the organization was wrapped around its spokes discussing how to proceed.

The team had provided a starting point, a prototype that validated the approach and could be built upon. The team quickly gained formal status and funding, and eventually came up with a solution for legacy database access.

By using Scrum, the team was able to cut through the noise and start delivering valuable product. Time that would have otherwise been wasted was spent working. The team was able to focus itself and deliver product. Management was able to help the team stay focused. The team continued for another year, building a general-purpose middleware business object server with access to specific databases. The team members became consultants to other organizations that used the middleware. As they consulted, they spread Scrum.

In the next sections, I'll describe the details of the Scrum practices I implemented in this case study so that you, also, can implement Scrum and manage Scrum projects.