Agile Software Development with Scrum
CHAPTER 1

Introduction

"In today's fast-paced, fiercely competitive world of commercial new product development, speed and flexibility are essential. Companies are increasingly realizing that the old, sequential approach to developing new products simply won't get the job done. Instead, companies in Japan and the United States are using a holistic method; as in rugby, the ball gets passed within the team as it moves as a unit up the field." (Reprinted by permission of Harvard Business Review From: "The New New Product Development Game" by Hirota Takeuchi and Ikujiro Nonaka, January, 1986. Copyright 1986 by the Harvard Business School Publishing Corporation, all rights reserved.)

This book presents a radically different approach to managing the systems development process. Scrum implements an empirical approach based in process control theory. The empirical approach reintroduces flexibility, adaptability, and productivity into systems development. We say "reintroduces" because much has been lost over the past twenty years.

This is a practical book that describes the experience we have had using Scrum to build systems. In this book, we use case studies to give you a feel for Scrum-based projects and management. We then lay out the underlying practices for your use in projects.

Chapters 5 and 6 of this book tell why Scrum works. The purpose of these chapters is to put an end to the ungrounded and contentious discussion regarding how best to build systems. Industrial process control theory is a proven body of knowledge that describes why Scrum works and other approaches are difficult and finally untenable. These chapters describe what process control theory has to say about systems development, and how Scrum arose from this discipline and theory. These chapters also lay out a terminology and framework from which empirical and adaptive approaches to systems development can ascend and flourish.

Scrum [Takeuchi and Nonaka], is a term that describes a type of product development process initially used in Japan. First used to describe hyper-productive development in 1987 by Ikujiro Nonaka and Hirota Takeuchi, Scrum refers to the strategy used in rugby for getting an out-of-play ball back into play. The name Scrum stuck because of the similarities between the game of rugby and the type of product development proscribed by Scrum. Both are adaptive, quick, self-organizing, and have few rests.

Building systems is hard and getting harder. Many projects are cancelled and more fail to deliver expected business value. Statistically,
the information technology industry hasn’t improved much despite efforts to make it more reliable and predictable. Several studies have found that about two-thirds of all projects substantially overrun their estimates [McConnell].

We find the complexity and urgency of requirements coupled with the rawness and instability of technology to be daunting. Highly motivated teams of highly skilled developers sometimes succeed, but where do you find them? If you are looking for a quick, direct way to resuscitate a troubled project, or if you are looking for a cost-effective way to succeed with new projects, try Scrum. Scrum can be started on just one project and will dramatically improve the project’s probability of success.

Scrum is a management and control process that cuts through complexity to focus on building software that meets business needs. Scrum is superimposed on top of and wraps existing engineering practices, development methodologies, or standards. Scrum has been used to wrap Extreme Programming. Management and teams are able to get their hands around the requirements and technologies, never let go, and deliver working software. Scrum starts producing working functionality within one month.

Scrum deals primarily at the level of the team. It enables people to work together effectively, and by doing so, it enables them to produce complex, sophisticated products. Scrum is a kind of social engineering aiming to achieve the fulfillment of all involved by fostering cooperation. Cooperation emerges as teams self-organize in incubators nurtured by management. Using Scrum, teams develop products incrementally and empirically. Teams are guided by their knowledge and experience, rather than by formally defined project plans. In almost every instance in which Scrum has been applied, exponential productivity gains have been realized.

As authors of Scrum, we have evolved and used Scrum as an effective alternative to traditional methodologies and processes. We’ve written this book to help you understand our thinking, share our experiences, and repeat the success within their own organizations.

In this book, we’ll be using the word “I” from now on rather than “we”, “Mike”, or “Ken”. Unless otherwise identified, “I” will hereafter refer to Mike Beedle in chapters 6 and 7, and to Ken Schwaber elsewhere.

1.1 Scrum At Work

The best way to begin to understand Scrum is to see it at work. After using Scrum to build commercial software products, I used Scrum to help other organizations build systems. The first organization where Scrum was tested and refined was Individual, Inc. in 1996.

Individual, Inc. was in trouble and its leaders hoped that Scrum could help them out. Individual, Inc. published an online news service called NewsPage. NewsPage was initially built using proprietary technology and was subsequently licensed to companies. With the advent of the Internet, Individual, Inc. began publishing Personal NewsPage as a website for individuals.

Eight highly skilled engineers constituted the Personal NewsPage (PNP) product development team. Though the team was among the best I’ve worked with, it suffered from a poor reputation within Individual, Inc. It was said the PNP team couldn’t produce anything, that it was a “total disaster.” This belief stemmed from the fact that there hadn’t been a new PNP release in nearly nine months. This was in 1996, when Internet time hadn’t yet taken hold of the industry, but nine months was already far too long. When I discussed this situation with marketing, product management, and sales, they said they couldn’t understand the problem. They would tell the PNP team what they wanted in no uncertain terms, but the functionality and features they requested never were delivered. When I discussed the situation with the disgruntled PNP team, it felt that it was never left alone to develop code. The engineers used the phrase “fire drill.” The team would think about how to deliver a required piece of functionality, start working on it, and it would suddenly be yanked off onto the next hot idea. Whenever the PNP team committed to a project, it didn’t have enough time to focus its attention before product management changed its mind, marketing told it to do something else, or sales got a great idea that had to be implemented immediately.

The situation was intolerable. Everyone was frustrated and at odds with each other. Competition was appearing on the horizon. I asked Rusty, the head of product management, to come up with a list of everything that people thought should be in PNP. He already had a list of his own and was reluctant to go to everyone and ask for his or her input. As he said, “If the PNP team can’t even build what we’re asking it to do now, why should we waste the effort to go through list building again?” However, Rusty did as I asked and compiled a comprehensive list. He also met with the PNP team to see if it knew of technology changes that needed to be made to implement the requirements. These were added to the list. He then prioritized the list. The PNP team gave development time estimates. Rusty sometimes changed priorities when it became apparent that items with major market impact didn’t take much effort, or when it became apparent that items with minor market impact would take much more effort than they were worth.

I asked Rusty to change the product requirements process. People currently went straight to the PNP team to ask for new product features and functionality. I thought it could be more productive if it only had one source of work and wasn’t interrupted. To implement this, Rusty suggested that people take their requests only to him. He added their requests to his list. He then reprioritized the list based on their presentation of the feature’s importance, his estimate (after talking to someone on the PNP
team) of how long it would take to implement, and the other work on the list. I advised Rusty to put every suggestion on the list. He never had to say “no.” Instead, he only had to prioritize. There were no “bad” ideas, just ideas that probably would never get implemented. Rusty advised everyone at Individual, Inc. that the PNP team would only schedule work for PNP based on his prioritized list of work. He started calling his list the Product Backlog list.

Rusty liked the Product Backlog list. He never had to finalize requirements for a product release. Instead, he just maintained a list of what was needed in the product, based on the best information available to him at the time. The list was always current and always visible. He kept the Product Backlog list on a spreadsheet on a public server, so everyone knew what was going into the product next. Another benefit was that the PNP team wasn’t interrupted as much. Individual, Inc. was a small company, where everyone knew each other. It had previously been hard to keep people from going straight to the PNP team with requests. Sometimes they would try approaching an engineering friend and asking for a favor: “Could you just sneak this one feature in, this once?” But Rusty insisted that all the engineers on the team stand firm. He became the keeper of the requirements, all listed by priority on the Product Backlog list.

I suggested that the company adopt a practice I had used previously: iterative, incremental development. I called each iteration a Sprint, and the results of the iteration were called a Product Increment. I suggested using Sprints so the PNP team would be left alone to focus on its task of building product functionality. It wouldn’t be asked over and over again, “How’s it going, are you ready to build the next thing? Have you implemented the last thing?” Sprints were intended to give the team control of its time and destiny. I also suggested a fixed duration for every Sprint. The PNP team asked for thirty days, which it felt would be enough time to build and test an increment of functionality. The team suggested putting each Product Increment into production on the web server at the end of every Sprint. The team was suggesting monthly Internet releases! All of my experience had been with shrink-wrap software, where a new release had to be distributed to all customers who then had to schedule its implementation. Since PNP was an Internet product, it only needed to be updated in one place for all users to realize the benefits. Of course, if the team put up something that didn’t work, all users would be immediately affected, so testing took on a new importance.

The PNP team met with Rusty to determine what to develop in the first Sprint. There was a lot of negotiation at this meeting, with much discussion about the details of the requirements and how to implement them. Some of the estimates changed as the implementation details were thought through. Some lower priority Product Backlog was included because it was essentially “free” once an area of code was opened for a higher priority backlog item. The meeting lasted all day. At the end of the meeting, the team had committed to implement a certain amount of the Product Backlog during the Sprint, and it had worked out a rough idea of the design and implementation details. Everyone knew what the PNP team was going to do for the next thirty days.

Of course, the PNP team was still approached innumerable times (even by Rusty) with requests to develop functionality that was not on the Product Backlog for the current Sprint. People who made these requests were asked to wait and to put these items in the Product Backlog. If their requests became top priority, they would be implemented in the next Sprint. Because the Sprint was only for thirty days, everyone could accept waiting until the end of the Sprint.

The PNP team worked without interruption during the Sprint, other than for product support and maintenance needs. PNP was still a little shaky because of the old quick and dirty process used to respond to “emergency” functionality requests. The existing policy was for the more junior engineers in technical support to make the fixes. However, these engineers were not familiar enough with the code to fix it. In order to solve this problem, I instituted the following policy: whoever writes code owns it forever. Although this detracted from the PNP team’s ability to fully focus on the Sprint work, it quickly improved the quality of the code.

During the Sprint, the PNP team questioned me about what engineering techniques had to be used, what type of documentation was required, and what design artifacts had to be created. Eventually, after consulting technical support, we all agreed that how the team did its work was up to them. This was especially the case since each team member owned the code that he or she wrote in perpetuity. However, I did stipulate the PNP team had to produce an updated product technical illustration with each Sprint (and release) that could be used to understand the product design and code.

At the end of the Sprint, the PNP team had implemented the functionality it had committed to — and more. Since it had an opportunity to focus on its work for the first time, it had accomplished more than expected. The team was ready to present the Sprint Product Increment, the new release. Rusty had the team demonstrate it to management and some of the customers. The audience was delighted, and immediately authorized the team to put the new functionality up on the production web server. After a drought of nine months, the PNP team had produced a new release within one month. The team went on to repeat this performance again and again. Before I left, it had generated another five releases.

While the PNP team was working on its first Sprint, the rest of the organization was still asking it to do “favors.” The team had a hard time...
whole team had to stay through the duration of the Daily Scrum to listen. Scrum had been brought in to increase the team’s productivity, but the Daily Scrum was starting to turn into a massive waste of the team’s time. I instituted some very simple practices to solve this problem and return the Daily Scrum to its initial intent. First, only the team members were allowed to talk. No one else could talk. If you weren’t on the team and you wanted to attend, you had to stay quiet. Second, the team was only allowed to talk about three things—what it had done since the last meeting, what it was planning on doing before the next meeting, and what was impeding its work. I called on the team members to report by going clockwise around the circle until everyone had reported.

As I ran the daily Scrum for the PNP team, it became apparent that I was fulfilling a management job. I bloated interference, allowed the team to keep focused, removed impediments and helped the team reach decisions quickly. This was a radical change, a flip, to what management had previously done. The team figured out how to do what it had committed to do. Management’s new and primary job was to maximize the team’s productivity, to be there to help it do the best that it could.

When I left Individual Inc., Scrum had been implemented in all three major product lines. At that time, Individual, Inc. went through a complete change in management, removing the founder and bringing in new people. Because of Scrum, though, the teams stayed focused and continued to regularly crank out new releases.

1.2 Quick Tour of Scrum

After Individual, Inc. I had a set of nomenclature and practices for Scrum. Let’s take a quick tour of this Scrum. Figure 1.1 shows the overview of the Scrum process.

Scrum is often used when a systems development project is being initiated. List all of the things that the system should include and address, including functionality, features, and technology. This list is called the Product Backlog. The Product Backlog is a prioritized list of all product requirements. Product backlog is never finalized. Rather, it emerges and evolves along with the product. Items that have high priority on the Product Backlog are the ones that are the most desired. Product backlog content can come from anywhere: users, customers, sales, marketing, customer service, and engineering can all submit items to the backlog. However, only the Product Owner can prioritize the backlog. The Product Owner effectively decides the order in which things are built.

Small, cross-functional teams perform all development (Scrum Teams). These teams take on as much Product Backlog as they think they can turn into an increment of product functionality within a thirty-day iteration, or Sprint. Every Sprint must finish by delivering new executable
Product
Backlog
Team
Capabilities
Business
Conditions
Technology
Stability
Executable
Product
Increment

FIGURE 1.2: Input for new Sprint

product functionality. Architecture and design emerge across multiple Sprints, rather than being developed completely during the first Sprints. See Figure 1.2 Input for new Sprint for an overview of how a new Sprint is formed.

Multiple teams can develop product increments in parallel, all teams working from the same Product Backlog. The Scrum Teams are self-organizing and fully autonomous. They are constrained only by the organization’s standards and conventions, and by the Product Backlog that they have selected. How the Product Backlog will be turned into a product increment is up to the team to decide. The team maintains a list of tasks to perform during each Sprint that is called a Sprint Backlog.

Scrum relies on team initiative and integrity. During the Sprint, a management representative (Scrum Master) enforces Scrum practices and helps the team to make decisions or acquire resources as needed. The team must not be disturbed or given direction by anyone outside of it while it is in a Sprint.

The Scrum Team meets daily for a short status meeting, called the Daily Scrum. At the Daily Scrum, progress is reviewed and impediments identified for removal by management. The Daily Scrum is an excellent place to observe how much progress a team is making.
At the end of the Sprint, the team gets together with management at a Sprint Review Meeting to inspect the product increment the team has built. They either build on what was developed, scavenge it, or throw it away. However, the pressure to build on what's been developed is high. The thirty day Sprint duration ensures that the worst that happens is that thirty days are lost should the team prove unable to develop any useful product functionality.

After the product increment is inspected, management often rearranges the Product Backlog to take advantage of what the team has accomplished. The Product Backlog has more meaning when viewed in light of the partially developed product. Sometimes so much product is built that management selects an earlier release schedule. In this case, the next Sprint can be used to release the product.

Once the Product Backlog has been stabilized, the team again selects top priority Product Backlog for the next Sprint. The team then goes through another iteration of work, pushing through another Sprint. This cycle continues until the product - based on Empirically Managing cost, time, functionality, and quality - is deemed potentially releasable. Release Sprints are then devised to bring the product to release-readiness.

Scrum is straightforward. By stripping away inappropriate and cumbersome management practices, Scrum leaves only the essence of work. Scrum leaves a team free to go to it, to work its heart out and build the best products possible. Although the Scrum process seems simple and skeletal, it provides all the necessary management and controls to focus developers and quickly build quality products.

1.3 Statements About Scrum

"The problem for engineers is that change translates into chaos, especially when a single error can potentially bring down an entire system. But, change also translates into opportunity. It’s as simple as this: if there is time to put a certain amount of functionality into the product easily, then there is time to put in more functionality at the price of a certain amount of disruption and risk. Thus does madness creep into our projects - we will tend to take on as much risk as we possibly can."

James Bach. (Courtesy of Cutter Information Corp.)

1.3.1 From Jeff Sutherland

Jeff invented many of the initial thoughts and practices for Scrum prior to formalizing and commercializing Scrum with Ken Schwaber. This is a retrospective on Scrum and its implementation in five companies.

Scrum was started for software teams at Essel Corporation in 1994 where I was VP of Object Technology. We built the first object-oriented design and analysis tool that incorporated round-trip engineering in the initial Scrum-based project. A second Scrum-based project implemented the first product to completely automate object-relational mapping in an enterprise development environment. I was assisted by two world-class developers, Jeff McKenna, now an Extreme Programming (XP) consultant, and John Scummtalades, now a development leader for object-oriented design tools at Rational Corporation.

In 1995, Essel was acquired by VMARK, and Scrum continued there until I joined Individual in 1996 as VP of Engineering. I asked Ken Schwaber to help me incorporate Scrum into Individual’s development process. In the same year I took Scrum to IDX when I assumed the positions of Senior VP of Engineering and Product Development and CTO. IDX, one of the largest healthcare software companies, was the proving ground for multiple-team Scrum implementations. At one point, I had over 600 developers working on tens of products. In 2000, I introduced Scrum to PatientKeeper, a mobile/wireless healthcare platform company where I became CTO. So I have experienced Scrum in five companies, with consulting assistance from Ken Schwaber in three of those companies. These companies varied widely in size and were proving grounds for Scrum in all phases of company growth: from startup, to initial IPO, to mid-size, and then to large company with a 30-year track record.

There were some key factors that influenced the introduction of Scrum at Essel Corporation. The book Wicked Problems, Righteous Solutions [DeGrace] reviewed the reasons why the waterfall approach to software development does not work today. Requirements are not fully understood before the project begins. The user knows what they want only after they see an initial version of the software. Requirements change during the software construction process. New tools and technologies make implementation strategies unpredictable. DeGrace and Stahl reviewed “All-at-Once” models of software development that uniquely fit object-oriented implementation of software.

The team-based “All-at-Once” model was based on the Japanese approach to new product development, Sashimi and Scrum. We were already using production prototyping to build software. It was implemented in slices (Sashimi) where an entire piece of fully integrated functionality worked at the end of an iteration. What intrigued us was Takeuchi and Nonaka’s description of the team building process in setting up and managing a Scrum [Takeuchi and Nonaka]. The idea of building a self-empowered team where everyone had the global view of the product being built seemed like the right idea. The approach to managing the team that had been so successful at Honda, Canon, and Fujitsu resonated with the systems thinking approach being promoted by Senge at MIT [Senge].
We were also impacted by recent publications in computer science. Peter Wegner at Brown University demonstrated that it was impossible to fully specify or test an interactive system designed to respond to external inputs, i.e. Wegner's Lemma [Wegner]. Here was mathematical proof that any process that assumed known inputs, like the waterfall method, was doomed to failure when building an object-oriented system. We were prodded into setting up the first Scrum meeting after reading Coplien's paper on Borland's development of Quattro Pro for Windows. The Quattro team delivered one million lines of C++ code in 31 months with a 4 person staff growing to 8 people later in the project. This was about 1000 lines of deliverable code per person per week, probably the most productive project ever documented. The team attained this level of productivity by intensive interaction in daily meetings with project management, product management, developers, documenters, and quality assurance staff.

Our daily meetings which we started at Easel were disciplined in the way we now understand as the Scrum pattern [ScrumPattern]. The most interesting effect in a Smalltalk development environment was “punctuated equilibrium”. A fully integrated component design environment leads to rapid evolution of a software system with emergent, adaptive properties resembling the process of punctuated equilibrium observed in biological species.

It is well understood in biological evolution that change occurs sharply at intervals separated by long periods of apparent stagnation, leading to the concept of punctuated equilibrium [Dennett]. Computer simulations of this phenomenon suggest that periods of equilibrium are actually periods of ongoing genetic change of an organism. The effects of that change are not apparent until several subsystems evolve in parallel to the point where they can work together to produce a dramatic external effect [Levy]. This punctuated equilibrium effect has been observed by teams working in a component-based environment with adequate business process engineering tools, and the Scrum development process accentuates the effect.

By having every member of the team see every day what every other team member was doing, we began to get comments from one developer that if he changed a few lines of code, he could eliminate days of work for another developer. This effect was so dramatic that the project accelerated to the point where it had to be slowed down. This hyper productive state was seen in several subsequent Scrums but never so dramatically as the one at Easel. It was a combination of the skill of the team, the flexibility of Smalltalk, and way we approached production prototypes that evolved into deliverable product.

A project domain can be viewed as a set of packages that will form a release. Packages are what the user perceives as pieces of functionality and they evolve out of work on topic areas. Topic areas are business object
components. Changes are introduced into the system by introducing a unit of work that alters a component. The unit of work in the initial Scrum was called a Synchstep.

System evolution proceeds in Synchsteps. After one or more Synchsteps have gone to completion and forced some refactoring throughout the system, or often simply provided new functionality to existing components, a new package of functionality emerges that is observable to the user. These Synchsteps are similar to genetic mutations. Typically, several interrelated components must mutate in concert to produce a significant new piece of functionality. And this new functionality appears as a "punctuated equilibrium" effect to builders of the system. For a period of time the system is stable with no new behavior. Then when a certain (somewhat unpredictable) Synchstep completes, the whole system pops up to a new level of functionality, often surprising the development team.

The key to entering a hyper productive state was not just the Scrum organizational pattern. We did constant component testing of topic areas, integration of packages, and refactoring of selected parts of the system. These activities have become key features of XP [Fowler].

Furthermore, in the hyper productive state, the initial Scrum entered the "zone". No matter what happened or what problems arose, the response of the team always was far better than the response of any individual. It reminded me of the stories about the Celtics basketball team at their peak, where they could do no wrong. The impact of entering the "zone" was not just hyper productivity. The personal lives of the people were changed. People said they would never forget working on such a project and they would always be looking for another experience like it. It induced open, team-oriented, fun-loving behavior in unexpected persons and eliminated those who were not productive from the team through peer embarrassment.

When Zassel Corporation was acquired by VMARK (now Informix), the original Scrum team continued their work on the same product. The VMARK senior management team was intrigued by Scrum and asked me to run a senior management team Scrum once a week to drive all the companies' products to the Internet. These meetings started in 1995 and within a few months, the team had caused the introduction of two new Internet products and repositioned leading current products as Internet applications. Some members of this team left VMARK to become innovators in emerging Internet companies. So Scrum had an early impact on the Internet.

In the spring of 1996, I returned to a company I co-founded as VP of Engineering. Ken Schwaber has documented much of the Scrum experience at Individual. The most impressive thing to me about Scrum at Individual was not that the team delivered two new Internet products in
a single quarter, and multiple releases of one of the products. It was the fact that Scrum eliminated about 8 hours a week of senior management meeting time starting the day the Scrum began. Because the company had just gone public at the beginning of the Internet explosion, there were multiple competing priorities and constant revision of market strategy. As a result, the development team was constantly changing priorities and unable to deliver product. And the management team was meeting almost daily to determine status of implementation of priorities that were viewed differently by every manager.

The solution was to force all decisions to occur in the daily Scrum meeting. If anyone wanted any status or wanted to influence any priority, they could only do it in the Scrum. I remember in the early phase, the SVP of Marketing sat in on every meeting for a couple of weeks sharing her desperate concern about meeting Internet deliverables and timetables. The effect on the team was not to immediately respond to her despair. Over a period of two weeks, the team self-organized around a plan to meet her priorities with achievable technical delivery dates. When she agreed to the plan, she no longer had to attend any Scrum or status meetings. The Scrum reported status on the web with green lights, yellow lights, and red lights for pieces of functionality. In this way the entire company knew status in real time, all the time.

During the summer of 1996, IDX Systems hired me away from Individual to be their SVP of Engineering and Product Development. I replaced the technical founder of the company who had led development for almost 30 years. IDX had over 4000 customers and was one of the largest healthcare software companies with hundreds of developers working on dozens of products. Here was an opportunity to extend Scrum to large-scale development.

The approach at IDX was to turn the entire development organization into an interlocking set of Scrum. Every part of the organization was team based, including the management team that included two vice presidents, a senior architect, and several directors. Front line Scrum teams met daily. A Scrum of Scrum that included the team leaders of each Scrum in a product line met weekly. The management Scrum met monthly.

The key learning at IDX was Scrum scales to any size. With dozens of teams in operation, the most difficult problem is ensuring the quality of the Scrum process in each team, particularly when the entire organization has to learn Scrum all at once. IDX was large enough to bring in leading productivity experts to monitor productivity on every project. While most teams were only able to meet the industry average in function points per month delivered, several teams moved into a hyper productive state producing deliverable functionality at 4-5 times the industry average. These teams became shining stars in the organization and examples for the rest of the organization to follow.

In early 2000, I joined PatientKeeper, Inc. as Chief Technology Officer and began introducing Scrum into a startup company. I was the 21st employee and we grew the development team from a dozen people to 45 people in six months. PatientKeeper deploys mobile devices in healthcare institutions to capture and process financial and clinical data. Server technology synchronizes the mobile devices and moves data to and from multiple backend legacy systems. A complex technical architecture provides enterprise application integration to hospital and clinical systems. Data is forward deployed from these systems in a PatientKeeper clinical repository. Server technologies migrate changes from our clinical repository to a cache and then to data storage on the mobile device. Scrum works equally well across technology implementations. The key learning at PatientKeeper has been around introduction of Extreme Programming (XP) techniques as a way to implement code delivered by a Scrum organization. While all teams seem to find it easy to implement a Scrum organizational process, they do not always find it easy to introduce new XP programming. We have been able to do some team programming and constant testing and refactoring, particularly as we have migrated all development to Java and XML. It has been more difficult to introduce these ideas when developers are working in C and C++, our legacy technology.

After introducing Scrum into five different companies with different sizes and different technologies, I can confidently say that Scrum works in any environment and can scale into programming in the large. In all cases, it will radically improve communication and delivery of working code. The next challenge for Scrum, in my view, is to provide a tight integration of the Scrum organization pattern and XP programming techniques. I believe this integration can generate more hyper productive Scrum teams on a predictable basis. The first Scrum did this intuitively before XP was born and that was its key to extreme performance and life changing experience. In addition, the participation of Scrum leaders in the Agile Alliance [Agile], which has absorbed all leaders of well-known lightweight development processes, will facilitate wider use of Scrum and its integration with extreme programming.

1.3.2 From Ken Schwaber

Ken developed and formalized the Scrum process for systems development.

My company, Advanced Development Methods (ADM), built and sold process management software in the early 1990's. Many IT organizations used ADM's product, MATE (Methods and Tool Expert), to automate their methodologies. For example, Cooper's & Lybrand used MATE to automate SUMMIT DTM, their systems development methodology, for both
internal use and use by their customers. IBM also used MATE, automating its outsourcing, software development, and change management methodologies with it. The methodologies that MATE automated for these companies were the traditional “heavy” methodologies.

In MATE’s heyday, the backlog of development work was daunting. Coopers & Lybrand and IBM were using MATE extensively, as were many of their customers. The number of requests for new functionality, new interfaces, and “nice-to-haves” was quite large. It was chaos! To help make sense of everything, I built a Product Backlog list; here I listed all requested functionality, planned technology, planned enhancements, and major bugs. I worked with our customers, including Coopers & Lybrand and IBM, to prioritize the list. However, the priorities never stayed still. They were always changing based on the most recent input from a customer or potential customer.

When I looked at everything ADM had to do, I was overwhelmed. I figured the next release was probably a year away. I realized that even if ADM achieved everything on Product Backlog, no one would be satisfied since requirements would have changed by the release date. In desperation, I started identifying product functionality that could be built in monthly cycles. At the end of the month, I’d review what had been built with the customers to see if I was on track. To my surprise, the customers were delighted with this approach. When they saw what ADM had built, they often changed their minds regarding their priorities. They often wanted to take immediate advantage of what was at hand, maybe with a few more tweaks.

I changed ADM’s development process to use two sequential cycles of one month each. In the first cycle, ADM would build functionality. In the second cycle, ADM would prepare it for release. If any engineers had extra time during the release cycle, they’d work on adding more to the next release cycle. The two-cycle approach seemed simple enough, but it had ramifications on the development and release environments. ADM really had to have the code management and release management systems and procedures thoroughly in place for this to work. Daily builds became a necessity. The rapid release cycle forced me to significantly upgrade all of the engineering practices. As a result, ADM became very efficient.

ADM built MATE using object-oriented technology (OO). In 1993, Jeff Sutherland, an active member in the Object Management Group (OMG) and the various OOPSLA SIG’s (and a good friend) asked what methodology was used by ADM. Jeff was sure that ADM was using one of its client’s methodologies. I still remember the look on Jeff’s face when I told him, “None — if ADM used any of them, it would be out of business.”

Jeff wanted to know what methodology ADM used because he was impressed by the short duration of ADM’s development cycles and the frequent releases of MATE. He wanted to understand so he could make OO similarly productive. At that time, OO wasn’t delivering the productivity that its proponents had initially promised and criticism was mounting.

Jeff had been head of engineering and development at Easel and then VMARK (both major software product vendors) and had long used and advocated OO. Jeff had also read about a new product development philosophy called Scrum. Jeff had implemented his interpretation of it at both Easel and VMARK. He hoped that my experience with the major methodologies would help him further formulate Scrum.

That conversation with Jeff was the beginning of a joint effort to formalize Scrum. We read everything that we could get our hands on, sometimes surprising ourselves when other fields such as complexity theory added to our understanding. We saw corollaries in chaos and complexity theory, recognized the beauty of emergent processes, and gained a better understanding of self-organization. We were profoundly influenced by research regarding software development practices at Borland and Microsoft. Of particular influence was their fierce focus on code. The final research occurred at DuPont’s Experimental Station in Wilmington, Delaware, where experts in process control theory reviewed our work and provided the theoretical foundations for Scrum.

Jeff and I worked together to create a formal description of the Scrum process. Scrum went from being a collection of thoughts that Jeff and I posted on our websites to a development methodology that I presented at OOPSLA’96. Since then, Scrum has become a major alternative to classic product development approaches. It has been adopted both by managers who wanted to ensure they got the best product they could, and by engineers who wanted to ensure they would be able to do their best work.

Since its introduction, Scrum has been used in thousands of projects worldwide. Mike, Jeff and I have personally implemented Scrum in hundreds of projects, and we’ve worked with and advised others in the United States, Europe, Australia, New Zealand, Singapore, the Philippines, Hong Kong, Ethiopia, and Indonesia. Scrum has been implemented for single projects as well as an organizational product development process. Scrum has been used for such diverse, complex efforts as realigning a product and customer base for a new Y2K compliant release, preparing a tunable laser subsystem for fiber optic networks for a trade show, and rapidly creating advanced teleradiology systems.

1.3.3 From Mike Beedle

Mike has long been a Scrum innovator and practitioner. Mike recently has wrapped Extreme Programming engineering practices with Scrum.
Chapter 1. How the Book Is Organized

As an experiment, do this: take three programmers and give them a project and a room. To begin the project, they will talk to the customer and find out what the customer wants and what is important to him or her. Together with the customer they will create a prioritized “feature list” – in Scrum we call this a Product Backlog. To actually implement something, they will meet with the customer and choose some features to implement first based on the priority of the features – in Scrum we call the meeting a Sprint Planning Meeting and the list of items to be implemented the Sprint Backlog. As they develop the software, they will run into issues and add them to their iteration “to do” list (i.e. their Sprint Backlog). To see where everyone is, they will have informal meetings to tell each other what they are working on, what issues they have and what they will be working on next – in Scrum we call these meetings Daily Scrums. As they implement features they will show their management and their customer how things look – in Scrum we call this Sprint Review Meeting.

However, if managing a software project is so simple, why do we have thousands of volumes of project management without this information? If these activities are so natural and common sense, why is this knowledge not explicit? If these activities are taking place despite the method used when deliveries to production take place, why are they not documented? If these are the steps that really drive a project, why do we waste our time with other things?

This is why I am writing this book. This information is first and foremost, important, but it is also the most natural, simple, and common sense way to manage a software project. Somewhere along the line, we forgot this basic and instinctive way of managing software projects, but unfortunately, we pay a very high price for this memory loss.

I cordially invite you to try Scrum. You won’t regret it.

1.4 How the Book Is Organized

Chapters 2, 3, and 4 are for the reader that wants to understand Scrum and apply it to a project or organization.

Chapter 2 (Get Ready for Scrum!) provides an overview of how Scrum is different, why it works, and what it feels like in a Scrum project.

Chapter 3 (Scrum Practices) is a step-by-step description of how Scrum works. Follow the instructions in this chapter and you’ll be using Scrum.

Chapter 4 (Applying Scrum) defines management principles and practices for building products and systems with Scrum. Iterative, incremental systems development requires empirical management practices. This chapter shows how to use these practices.

Chapters 5 and 6 provide the theoretical underpinnings for Scrum and other similar system and product development practices. These chapters are for the advanced reader.