Chapter 1

Adapting to Scrum

Why Consider Using Scrum?

Dissatisfaction is essential for change. That's an established truism in the consulting businesses, where consultants know that their opinions and evaluations won't be taken seriously and no meaningful change can happen unless they can first establish that dissatisfaction with the status quo is sufficient to justify taking real action. But change is frightening. So why do we need a new method for building software? Because the traditional way of building software is even more frightening.

Clients are accustomed to huge losses; missed deadlines are the norm; more software projects fail than succeed. In such a climate, something new must evolve, and agile development and management principles and methods have evolved to satisfy the need. I know the situation all too well.

A decade ago, I began my foray into software outsourcing as a client of a large intranet system for a publicly listed holding company. The system was to link all the employees of the 34 companies owned by this holding company so that they could share and retain knowledge. I learned firsthand through that experience how challenging and frightening software outsourcing can be. The fear stems from the fact that the process is out of the client's control. A client can make requests and even demands, but ultimately the quality of the software, the time it is delivered, and the total cost often feel and even are completely beyond the client's control. Even if they have a rock-solid contract stipulating delivery dates and a fixed price, they know that there's something the contract can't cover. Perhaps it's the ongoing maintenance costs, maybe it's a feature that wasn't described just right in the documentation, or perhaps it's that killer feature that no one even thought of while defining the project. Something key is always out of the client's control.

Agile development processes, and the Scrum management framework in particular, restore a sense of control and place the responsibility for the various aspects of the software development process, scope, time, cost, and quality, in the hands of the stakeholders who are best able to manage them.

Scope: The entire set of features, performance requirements, usability requirements and any other requirements that are necessary to consider the project complete and successful.

What is Agile Development?

Discipline is the essence of agile development. That might strike some as odd, given that many people still think of agile development as unplanned and without documentation. To anyone who has done it, though, the one thing that leaves the biggest impression is that in reality, the value of agile development comes only when the team achieves a level of discipline rarely experienced in non-agile projects. The reason is, that lacking other control measures and having as it does total transparency to stakeholders, an agile team is under tremendous pressure to perform, delivering fully-working and tested features fast without ever allowing quality to slacken.

Agile development practices achieve the goal of producing quality software fast by prioritizing first quality practices such as continuous integration, test-driven development, concurrent testing, pair programming, and shared code ownership (more on these later) while at the same time, making progress very visible to the product owner and other stakeholders. It changes the focus of the development team from scope first, then time and lastly quality (so typical on traditional projects) to quality first, followed by scope and time. Scope is kept small (only fixed for the current iteration of 1-4 weeks, typically) so that the project can be highly adaptable to changing requirements while still giving the team the stability of an unchanging near-term goal.

In a nutshell, agile development is all about delivering high-quality working product fast to achieve maximum value as early as possible without sacrificing maintainability.

Who Invented Scrum?

The Scrum framework for software development, like many software methodologies, evolved from lessons learned in manufacturing. It was first named in 1986 in an article on manufacturing that appeared in Harvard Business Review (add citation Jan 1986). This article compared the new manufacturing approach to rugby, because the whole team strives to proceed together. Its first documented use in software development was in 1993, and in 1995 Ken Schwaber formalized the rules of Scrum for software development. Scrum exploded onto the mainstream in 2001 with the publication of "Agile Software Development with Scrum" by Schwaber and Mike Beedle. At the time that I adopted Scrum, this was the only text available in print to use as a guideline.

Who Uses Scrum?
Scrum is not just a great process for building software. I've used it for planning a conference and for organizing internal marketing campaigns, but this book is about offshore software development. Scrum has been successfully used on projects ranging from very small with teams of just a few people to very large with teams of over a hundred.

Entrepreneurs, bootstrapping start-ups, small businesses, and global enterprises use Scrum to improve quality, efficiency and control. Scrum is ideal for in-house software development, but the practices can also be applied to manufacturing, event planning, and many other types of team-based project work. Scrum is extremely valuable when outsourcing and offshoring, because it gives the client a degree of insight into the development process that not only serves to allay very reasonable concerns about quality control, progress, and mutual understanding of requirements.

In my personal experience I have coached Scrum teams ranging in size from two to fifteen people, sometimes co-located and other times spread over three continents. My clients have ranged from one-person unfunded start-ups to major multinational corporations and non-governmental organizations, including the United Nations.

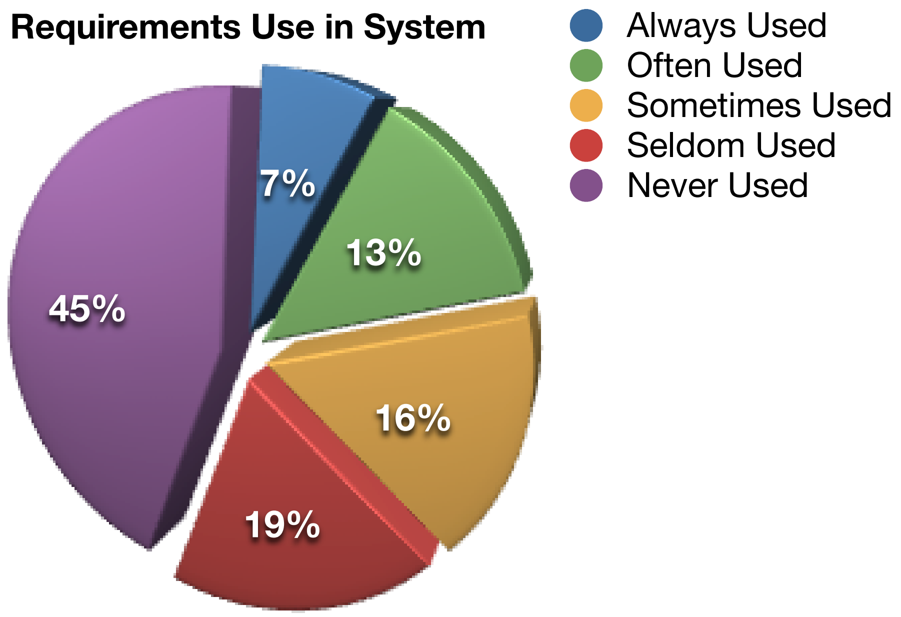
Why does Scrum work?
Scope changes are one of the primary causes of stress in traditional software projects. Agile approaches to software development eliminate scope change as a problem by recognizing its inevitability. In the past six years since I adopted Scrum as my preferred project management framework I have never built a product to the original spec. Inevitably, in the process of working closely with clients and delivering working software fast, I see that they get their best ideas when they have their hands on their product and when the get stakeholder feedback early.

Most software development contracts try to define the scope of the work up front, and place all of the responsibility for managing the scope on the development team. That arrangement creates problems in light of the experiences that I've just related. When a contract defines a fixed scope of work and usually a fixed budget, too, there is only one thing that can give if both are to be met, and that's quality. It doesn't take a genius to know that when the chips are down and the deadline nears, as costs begin to exceed the budget and a development team is facing the very real possibility of taking a loss on a project by going over a fixed budget, desperate measures like working people overtime, cutting testing, and substituting lower-cost interns for senior programmers become not just tempting, but even a survival imperative.

What's more, fixed-scope agreements pit the development team or project manager against the client by incentivizing them differently. The client has a strong incentive to ensure that the scope is interpreted in the broadest possible sense, even to the extent of slipping in new features for "free." The software team has the opposite incentive, because the more narrow the scope interpretation, the higher the profits on a fixed-price contract. When client and service provider are adversaries, it's hard to establish a working, trusting relationship.

Agile processes like Scrum work by giving each party to a contract responsibility over that aspect of the project that they are best suited to control. The development team is responsible for delivering a quality product as efficiently as possible. The client is responsible for controlling the scope of work, which also controls the budget. In this way, both parties can collaborate without friction to produce the product the customer needs, adapting as they go to changing requirements, without sacrificing quality.

Statistics produced by the two CHAOS studies conducted by the Standish Group revealed, among other very useful observations, that the majority of features in software were rarely or never used [citation]. It's not hard to imagine that if you remove over 60% of the features from a product, you'll get a lower-cost and higher-quality product. The costs will be lower because, of course, there will be less work involved in coding and testing the smaller product. The quality will be higher because, most obviously since bugs reside in code, less code equals less bugs. Less obviously, lower complexity leads to more maintainable and more useable software. Maintainability and usability are important quality measures that are often overlooked. It doesn't really matter if the software is "bug-free" if no one can use it and it costs a fortune to maintain.



In traditional software projects, the scope of the product tends to get expanded beyond all reasonable proportions because it is designed by committee. The is a strong incentive on the management team to think of everything they might want the software to do, because they know that changing the feature set after project initiation will require intense re-negotiations and change requests. So the rack their brains to imagine everything that any user might desire. The result is what the Standish Group found, software in which 63% of the features weren't appreciated by almost all users, but which had to be planned, coded, tested, and paid for.

By allowing the client to change and adapt the feature set as software is developed and delivered, agile approaches like Scrum remove the desire to created bloated products by building in only what users actually want, thus reducing pressure on the budget and allowing the development team to focus on creating the highest-quality product that are capable of producing.

[Insert explanation of What is Software Quality - thanks @amrk]

What does it mean to "embrace change"?
Change is an inevitability in software development. I've managed, worked on, or consulted with hundreds of software projects and I know that the myth of stable requirements lives only in the minds of managers. The guys on the ground know that requirements change. Anyone who's been coding and had a stakeholder lean over their shoulder and "adjust" their priorities knows that's awkward and irritating. The reason why is that there is a significant mental cost involved in task-switching. It's a cost that good programmers resent because it lowers their productivity and good programmers are proud of their productivity.

So does agile development with its mantra of embracing change mean that good programmers grin through clenched teeth and smilingly accept change in the middle of a task? Of course not. One of the great beauties of agile development is that all agile methodologies and frameworks have the notion of the time-boxed iteration. That means that there is a short period of time during which requirements are static. It's an elegant solution for both the stakeholders and the development team. It means that the programmers, designers, and testers can start work every day with confidence that their plans for the day and for the following days won't be interrupted, but product owners, mangers, stakeholders giving input into the product also know that when they have a new or different idea, they can adjust the backlog for future iterations and their changes will be implemented soon, because iterations are short. In this way, everyone wins. The software adapts elegantly to changing requirements and the development team is free to focus on the most efficient way to maximize quality and productivity.

Confrontation is the enemy of collaboration. When I managed waterfall or waterfall-like projects in the early days of my career, a big part of my job was confronting clients with the costs of their demands. New ideas and improvements were anathema, because they meant rethinking, rework, re-estimating, and re-negotiating deadlines and budgets. Telling clients "no" was the worst part of my job. That is perhaps the primary reason that I adopted Scrum six years ago. It is empowering to be able to work with a customer to stimulate their creativity and add value to the final product. I also relished the idea of sustainable successes, in cooperation with customers. Now, years later, I've realized a previously impossible dream. I've never failed.

Again, citing the CHAOS study, most software projects fail. Failure is the norm in my industry. Those that don't fail outright, in the sense that they are never completed, go over budget and miss deadlines. Since adopting and continually improving my Scrum process, I have never failed to deliver software that satisfied a client, and out of dozens of web applications, only three have gone over budget and two have missed deadlines. In those cases, it was almost always the client's decision to invest more time, rather than a failure of planning or execution. Those kind of results would have been unthinkable to me a decade ago.

There are many approaches to agile adoption, and all require the support of the entire organization, but what worked for me was simply trying. Indeed, when you have a process in place I believe that an experienced trainer or coach is always a boon, but I do not agree with those who suggest that the risk of failing to implement a new process to perfection the first time necessitates the participation of an agile coach. Just the opposite. In order to have a real sense of ownership in the process, the team should be involved in the design, implementation, and critical review of their own process. I began by reading a few books, following agile blogs [todo add a footnote with some link love for my favorite bloggers and authors], and getting the general notion that there are a few key components to an agile process:

- Focus on quality rather than scope
- Timeboxed iterations
- Self-organizing teams
- Frequent working releases
- A Flexible backlog of features
- Concurrent testing
- Periodic process reviews

That was enough to get me started on the right foot. Not only were we quickly delivering better software on time, but we had a feedback loop in place to ensure consistent improvement. I'll get into the details of implementing these core processes in later chapters, but put simply, my opinion is that if the team is talking to the product owner daily, releasing working and tested features on time in timeboxed iterations, and reviewing and improving the process, you have taken a huge step in the direction of sustainable success.

Scrum is deceptively simple to learn. The core practices are clear and easy to implement. When I began using Scrum in 2005 I had merely read the book Agile Software Development with Scrum and several blogs and launched straight into it. The real challenges of using scrum fall into three categories: discipline, continual improvement, and teamwork. Because there is a difference between using Scrum and excelling in Scrum; and there's no such thing as perfecting Scrum. A commitment to continual improvement by holding retrospectives at the end of every iteration and committing to adapting your practices for incremental improvement is the most sustainable route to excellence I've found. Years on, our practices still include all of the scrum practices we began using, but the ScrumMaster I was in 2005 would not recognize the process that we are using today. And my team is still learning with every iteration we complete.

Discipline is crucial to any agile development process, and so I have decided to dedicate an entire chapter to the question of why discipline is essential and how to cultivate it.

[Insert somewhere - What is discipline? (a whole chapter, perhaps?) - thanks, @MajorNichols]

Teamwork and especially the notion of a self-organizing team is a not nearly as simple as it sounds. Project managers have to learn to be team players, not team leaders. Senior programmers have to learn to share responsibility for code quality. The team has to learn to settle its own professional and interpersonal issues by itself. This is not a natural way of working in most companies, and so it's a matter that requires constant vigilance and careful consideration by everyone on the team. In some teams, it may be a specific component of the retrospective and it is also an area for continual improvement.

Nothing has changed my professional life like the switch to the Scrum framework. As a project manager in traditional teams, my job was rarely fun. It usually involved creating reams of documentation for features that would never be developed, negotiating change requests with clients and "pushing back" on perfectly good ideas, badgering programmers for updates and trying to encourage them to work faster than was really realistic to meet targets that had been negotiated before anyone fully understood the risks, and then trying to convince clients that the end result was what they asked for, even if it wasn't what they really needed. No fun at all.

That all changed with Scrum. Now I engage with clients and they become part of the team. All of my efforts are put into real value-adding activities. Programmers on my teams are happy, clients are happy, testers are happy. Everyone is happy almost all the time, because they feel in control of their lives. They know their roles on the projects, they understand their responsibilities, and they can see how they fit into the process. That is very close to how Aristotle defines happiness in his Nichomachean Ethics, knowing and fulfilling your role in life, and to the stoic idea of controlling what you can and accepting what you can't. These ideas are thousands of years old, but it wasn't until recently that they were applied to software development, and I feel very lucky to have started my career at the just right time to benefit from these new tools.

It's selfish of me to feel that the best impact of Scrum on my life was that I have great relationships with happy clients. As overwhelming a change as that is, I think it should properly be eclipsed by that fact that over the years and dozens of project, none has failed. A couple went over budget, but not to disastrous degrees. Every single one launched successfully. If you are not part of the software world you might be forgiven for saying, "Well, that's your job. Don't be so proud of not failing." Anyone in the software world with any experience realizes that failure is the norm. Most software projects fail. Those that don't invariably go dramatically over budget and blow deadlines out of the water. If you still don't believe me, I'll refer you again to one of the most comprehensive studies of software development practices ever done, the Standish Group's CHAOS report. The Standish Group found that within their sample, 31.1% of software projects never made it to completion. 52.7% cost over 189% of the estimated budget. Only 16.2% of all software projects in the study were completed on-time and on-budget. KPMG Canada did a study with similar findings, revealing that over 61% of the projects surveyed were considered a failure. The OASIG study done in the UK found that 7 out of 10 projects fail in some way. It's a wild world we live in, and for me, Scrum has tamed it.

[note: Citations for the studies above]

How do clients feel about the Scrum experience?
Empowered clients are easier to work with because they don't suffer from the feeling that they have to tread lightly because they depend on you. They are working with you because the want to. Giving them the tools they need to track and manage their own projects, or even to easily take the work away from you, means that they have a better experience and you do, too. Lack of control and dependance can lead to resentment, even when everything is going well. I've often marveled that a client whose project is going perfectly but who feels alienated from the process by lack of control and be more agitated and upset than a client who is actively involved in working with a team to solve a crisis.

Some of my clients were skeptical at first, but within an iteration or two, without fail, they are 100% on board with the scrum approach to software development, especially if they have never encountered agile processes before. The enhanced communication, access to the team, visibility into the process, control over the future, and the ability to be a value-adding contributer dramatically changes the dynamic of the relationship between client and vendor. I can't imaging going back to the old way of doing things.

One of the best aspects of agile development is that agile teams release early and often. This practice quickly satisfies stakeholders by showing them early that the team is productive. It also eliminates compounding any misunderstandings about requirements, catching mistakes early when they are easy and cheap to correct.

Perhaps the best thing about iterative releases is that in my experience customers get their best ideas only when they actually are using their software. If you wait until the product is finished to gather input on it from clients and stakeholders, your best opportunities to adapt to their real needs are wasted. What's more, as we saw from the CHAOS report, most of what you've built was wasted effort.

Customers who haven't experienced Scrum are often put off by the idea that they are required to participate daily or nearly daily. I find, though, that over the life of an agile project, the product owner spends less time using the scrum method then they typically spend in the planning phase using traditional waterfall methods. But because the work is spread over the life of the project, it is less intrusive, and more valuable.

In latter chapters, I'll talk about exactly what the product owner does in each phase of the product life cycle and I'll talk about the time commitments that are required based on my experience.

The most important message I want to convey is that Scrum provides a strategy for empowering everyone on the team in ways they might never have expected. Programmers are empowered to focus on quality and to choose the most effective ways in which to work. Project managers are empowered by a process which takes from their shoulders the burden of responsibilities for things that are beyond their control. It pulls them out of the spreadsheets and puts them in the thick of the action, working with people rather than just with numbers to actually contribute to a learning organization. And it empowers product owners by giving them total transparency into the development process and by allowing them to change their mind and contribute to the evolution of their product. It's just a more fun, more effective, and better way for a group of talented, professional people to create a new software product.

[General note: add more examples]