The Scrum Papers:
Nut, Bolts, and Origins of an Agile Framework

Jeff Sutherland and Ken Schwaber
Co-Creators of Scrum
Draft 29 Jan 2011, Paris

Jeff Sutherland, Ph.D.
CEO, Scrum, Inc.
One Broadway, 14th Floor
Cambridge, MA 02142
jeff.sutherland@scruminc.com

Dedication

This book is dedicated to the Godfathers of Scrum, Takeuchi and Nonaka [1], who gave Scrum its name and helped created a global transformation of software development. In 2011 Scrum is used in over 75% of Agile implementations worldwide. Many others have contributed to the creation of Scrum:

- Jim Coplien and the ATT Bell Labs Pasteur Project for the paper on the most productive software development team ever documented – the Borland Quattro Pro Project [2]. The first Scrum team implemented the Scrum daily meeting after reading this paper.
- Nobel Laureates Muhammad Yunus and the Grameen Bank for originating microenterprise development and the Accion International President's Advisory Board, responsible for much of microenterprise development in the western hemisphere. The strategy for bootstrapping the poor out of poverty has been a model for freeing hundreds of thousands of software developers from developer abuse caused by poor management practices.
- Alan Kay and his team at Xerox Parc [3] for inventing Smalltalk, the mouse, the graphical user interface, the personal computer, the Ethernet, and the laser printer. Listening to his insights on innovation inspired the first Scrum team to go from "good" to "great" [4].
- Professor Rodney Brooks for launching the startup now known as iRobot in space leased from Jeff Sutherland. He taught us the subsumption architecture [5], how to create simple rules to produce highly intelligent performance from complex adaptive systems.
- Christopher Langton of Los Alamos Labs and the Sante Fe Institute for coining the term "artificial life" and showing that increasing degrees of freedom up to the edge of chaotic behavior accelerated their evolution [6]. Scrum feels "chaotic" by intent, so as to accelerate software evolution.
- The French object-database developers working near the MIT campus at Graphael (later Object Databases, then Matisse Software) for demonstrating first in Lisp and then in C++ the Agile patterns of pair programming, radical refactoring, continuous integration, common ownership of code, world class user interface design, and other tips and tricks which Kent Bent used to create eXtreme Programming a decade later. These were all incorporated into the first Scrum.
- The Creative Initiative Foundation for their work with Silicon Valley volunteers to help make the world a better place, the underlying motivation driving the founders of Scrum. This connected the Co-Creators of Scrum with the early systems thinking of MIT Professor Peter Senge who later wrote "The Fifth Discipline."
- Capers Jones and his productivity experts at Software Productivity Research who analyzed and reanalyzed the output of early Scrum teams, as well as many of the software products built with Scrum during 1994-2000 [7]. These analyses allowed us to provide a money back guarantee that users would double productivity during the first month using tools created by the first Scrum.
- The first Scrum team – John Scumniotales (ScrumMaster), Don Roedner (Product Owner), Jeff McKenna (Senior Consultant), Joe Kinsella (object-relational mapping), Laurel Ginder (QA), and three Danish developers - Grzegorz Ciepiel, Bent Illum, and John Lindgreen. They endured repeated failure, depressing analysis of these failures in

front of their technical peers from other companies, and transcendence of their missteps. They were the first Scrum team to achieve the hyperproductive state for which Scrum was designed and their product, Object Studio, was reported as industry leader by computer trade journals. Little did they know that Scrum would be their greatest contribution.

- PatientKeeper, Inc., the first company to fully implement an "All at Once" or Type C Scrum involving the entire company in Scrum practice. This innovation in process design has been documented by Mary and Tom Poppendieck in their book on Lean Software Development [8]. "I find that the vast majority of organizations are still trying to do too much stuff, and thus find themselves thrashing. The only organization I know of which has really solved this is Patient Keeper [9]." PatientKeeper was the first company to achieve a hyperproductive revenue state driven by Scrum in 2007. Revenue quadrupled from 13M to 50M in one year.
- Last, but not least, many Scrum practitioners experience the ***quality without a name*** (QWAN) - a phrase used by Christopher Alexander in his book "The Timeless Way of Building" [10]. Alexander describes a certain quality that we seek, but which cannot be named. This may be the most important feature of Scrum and can only be spoken of as a set of core values - openness, focus, commitment, courage, and respect. It could be viewed as the "speed of trust" or one of the sources of "ba" often seen on Scrum teams. Ba is the Japanese term for the creative flow of innovation described by Takeuchi and Nonaka [11].

Thanks to the reviewers of the text who include among many others:

- Tom Poppendieck
- Henrick Kniberg
- Rowan Bunning
- Clifford Thompson

Introduction

Scrum derives from complex adaptive systems theory and was influenced by best practices in Japanese industry, particularly by lean development principles [12] implemented at companies like Toyota [13] and Honda [14], and knowledge management strategies developed by Takeuchi and Nonaka [11], now at the Hitotsubashi Business School in Japan, and Peter Senge [15] at MIT. It was enhanced by the patterns movement which evolved out of the Pasteur Project at ATT Bell Labs led by Jim Coplien.

Scrum is not a development method or a formal process, rather it is a compression algorithm for worldwide best practices observed in over 50 years of software development. The Scrum framework is simple to implement and automatically unpacks and encourages a software development team to deploy best practices documented in *Organizational Patterns of Agile Development*. Author, Jim Coplien, comments, "Scrum encapsulates 33 of the 45 patterns in my book. It takes two minutes for me to explain Scrum and over 60 pages to explain the patterns. This compression of best practices is an amazing characteristic of Scrum."

Another unusual aspect of Scrum is that it works in any domain. Jeff Sutherland is a coach to OpenView Venture Partners and they run their investment practice using Scrum with daily standup meetings. Jeff coaches religious organizations who find that Scrum radically improves their programs, finances, and new membership. Scrum has been used in companies across business domains and even for individual families in planning weddings, family chores, and children's schedules. It is a significant innovation in the way to get things done faster with higher quality while making the work experience more rewarding for all participants.

Scrum is used as an Agile practice that delivers software to end users faster, better, and cooler [16, 17]. As the Chief Product Owner at Yahoo observed, coolness is a requirement at Google, Yahoo, and most software game companies. Scrum supports a creative approach to development of complex and innovative systems and it scales to large numbers of developers. It is used on some of the world's largest projects at companies like British Telecom or Siemens because of high productivity with large distributed and outsourced development teams. It is the only software development process that has demonstrated linearly scalable when adding resources to large projects [18, 19]. In a properly implemented Scrum, productivity per developer stays the same when adding resources, a phenomena never seen before in software development.

The most profitable software product ever created (Google Adwords [20]) is powered by Scrum and the most productive large project with over a million lines of code (SirsiDynix [18]) used a distributed, outsourced Scrum implementation. CMMI Level 5 companies cut costs in half with Scrum while simultaneously improving quality, customer satisfaction, and the developer experience (Systematic Software Engineering [19]). At the same time, Scrum remains the process of choice in small entrepreneurial companies where it has its roots. OpenView Venture Partners in Boston invests only in Agile organizations and Scrum is the core process used in their portfolio companies.

The first software Scrum was created at Easel Corporation [21] in 1993 based on extensive research on successful projects worldwide, a deep analysis of the computer science literature, close collaboration with leading productivity experts, and decades of experience with advanced software technologies. Jeff Sutherland was the Chief Engineer for the Object Studio team that defined Scrum roles, hired the first Product Owner and ScrumMaster, developed the first Product Backlog and Sprint Backlog and built the first portfolio of products created with Scrum.

In 1995, Jeff introduced the Scrum team to Ken Schwaber, CEO of Advanced Development Methods. Ken agreed that Scrum was a better way to build software than traditional methods used at IBM and the big eight consulting firms and worked with Jeff to formalize the Scrum development process at OOPSLA'95 [22]. In the same year, Sutherland provided support for development of eXtreme Programming [23] by giving Kent Beck all background information on the creation of Scrum [24] and the results of two years of product development with the Scrum process from 1993-95. XP engineering practices evolved along with Scrum and the two leading Agile development processes work well together. Scrum and XP are the most widely used Agile practices worldwide and their creators are signatories of the Agile Manifesto.

Agile development is used globally as the best way to develop, maintain, and support software systems. Several papers on the early implementation of Scrum are of general interest. Later papers provide some of the nuts, bolts, and best practices of Scrum implementations. The design and implementation of an All-at-Once Scrum (Type C Scrum) at PatientKeeper to enable enterprise agility has been emulated by innovative companies in many countries. Case studies of CMMI Level 5 Scrum implementations and hyperproductive distributed, outsourced teams are of particular interest. In this book, knowledge gained from these studies are organized into a single volume to be readily accessible.

Scrum has made its way through the Pattern Languages of Programming Design (PLoP) process. Both Scrum and eXtreme Programming were affected by the software patterns movement and Mike Beedle, a Scrum signatory of the Agile Manifesto, led the effort to formally codify Scrum as an organizational pattern. His work published in Volume 4 of Pattern Languages of Program Design [25] in included here.

Scrum is designed to add energy, focus, clarity, and transparency to project planning and implementation. It will consistently:

- Increase speed of development
- Align individual and corporate objectives
- Create a culture driven by performance
- Support shareholder value creation
- Achieve stable and consistent communication of performance at all levels
- Enhance individual development and quality of life

The global expansion of Scrum in both the largest and smallest software companies and across all cultures is a testimony to the fact that Scrum delivers on its promise. While it is often said that Scrum is not a silver bullet, Scrum can be like a heat seeking missile when pointed in the right

direction. It's inspect and adapt approach to continuous quality improvement can do serious damage to outmoded business practices and many companies are now transforming entire organizations to take advantage of Scrum productivity, to delight customers, and to make the work environment better and more fun for development teams. It's focus on building communities of stakeholders, encouraging a better life for developers, and delivering extreme business value to customers, releases creativity and team spirit in practitioners and makes the world a better place to live and work.

For every member of the first Scrum team and for many teams that followed, individual lives were changed by Scrum. In Norway in 2008, the managing director of a Scrum company commented that annual revenue was quadrupled by Scrum. She said she was very happy about the success of the company but the best thing was that the people in the company felt so good about the way they worked together. She felt deeply grateful for the positive feeling of the people. So Scrum is as much about the heart as about the intellect and more about getting people to help one another than about project management. It can focus the spirit of the team and allow them to take their work to a higher level where everyone benefits in the process of delivering real value to each other, managers, customers, and end users.

Some people on the first Scrum team were concerned they would be searching for the rest of their lives to find another team that would provide a similar exhilarating feeling of working together to achieve great success. They feared they would never be able to find another opportunity and the first Scrum team would be the only work experience in their lives that deeply satisfied them. They cried in my office about this when the first Scrum company was acquired by a larger company and moved to a new location. Yet because of their generosity of spirit, their gift of blood, sweat, and tears, and the careful shepherding of Scrum by Ken Schwaber, they can relax. Scrum teams are found everywhere, from Silicon Valley to Katmandu. All people now have the opportunity to experience the benefits of Scrum.

Jeff Sutherland - Somerville, MA USA, 2010

Forward: Ikujiro Nonaka and The Scrum Way

by Jeff Sutherland, Paris, 2011

In 1993, the first Scrum team read "The New New Product Development Game" in the Harvard Business Review. The authors, Takeuchi and Nonaka, were visiting professors at Harvard when they published this paper in 1986. They described the best teams they had seen around the world as self-organizing teams characterized by autonomy, transcendence, and cross-fertilization. That is the team members were totally responsible, put the team first, and everyone learned how to do everyone else's job. The team leader had a facilitative style, helped the team translate outrageous senior management goals into practical implementation, and became a catalyst for innovation and organizational transformation.

Our first Scrum team had studied hundreds of papers and talked with many of the leading experts on software development and product management. The team had an outrageous goal ,to replace all the company's products with a brand new product in less than 6 months. No one could tell them what to do because no one knew what the product should be. It would be different and it would be ground breaking and we would figure it out. Nonaka's work was a life raft in a stormy ocean and we all grabbed it and held on.

Seventeen years later in 2011, I met Nonaka for the first time in Tokyo. He is a small man, a yoda-like figure who does not use a computer. Early in his career he worked with the military to figure out why the Japanese lost World War II. He is now one of the most influential management gurus on the planet and some say he has replaced Peter Drucker as the leading thinker in management innovation. His latest book, "Managing Flow: A Process Theory of the Knowledge-Based Firm" is based on a deep understanding of philosophy and human psychology. In it he elucidates the magic through which great products arise from the dynamic interaction of a team.

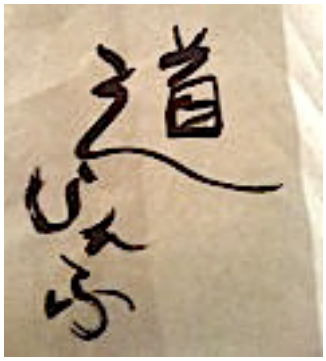
His primary concern in 2011 is to help Japan through its third great transition. The first was when the Shogun turned Japan over to the Emperor because once Admiral Perry showed up with guns, he could no longer maintain the lengthy effort to keep Japan isolated and impervious to western influences. The second was the aftermath of Hiroshima and the rebuilding of Japan. Now Japan has the highest per capita debt in the world. The growth of their economy is one of the lowest in Asia and the Chinese and the South Koreans are dramatic economic threats. The people all say there needs to be and there will be earth-shaking changes in Japan, and soon.

Nonaka knew nothing about software Scrum until about a year ago when an emerging Agile community in Japan attended Agile 2009. Kawaguchi Yasunobu and Kenji Hiranabe organized the first Agile conference in Japan and asked Nonaka to come and talk about Scrum. For their second conference, they invited me to meet Nonaka. who was surprised by Scrum and found it different from his meetings with the leading management and lean gurus. He said he felt humility from Scrum rather than the arrogance he had experienced elsewhere. He was so happy that his ideas that developed in a university setting had achieved such widespread practical implementation in the business world. He feels deeply that Japan needs radical innovation in this

critical time for his country and that Scrum can help. At dinner he said quietly, “Help us save Japan.”

I was deeply affected by Nonaka and by the dozen or so Japanese ScrumMasters who all want to become Scrum Trainers and are ready to do anything to achieve it, including learning English so they can appear before the review committee in America. During our product owner course, they kept asking me to draw a kanji character, repeatedly giving me parchment, a calligraphy pen, and an ink bottle. They said it was important that I practice. Not knowing what the character meant, I went along with them. Then late at night at dinner in a bar after everyone had half a dozen beers, they said, “Now is the time. You must draw the Kanji character for real now.” Though I drew a poor imitation of the excellent example they gave me to follow, they were quite happy with it. They said at last, “this means “The Scrum Way” and at the bottom is your signature.”

Suddenly all my years of training in Aikido dojos came back to me. Master Ueshiba, the founder, had drawn the kanji for “The Aikido Way” and I had studied under some of his direct disciples. The Way is a way of doing, a way of being, it is a way of life. And it requires unending practice in a community of practitioners, with relentless focus on continuous improvement. The “Way” kanji characters appear on the t-shirts, the kimonos, the publications, and doorways to the halls of those who practice the “Way.” I saw Scrum as if for the first time through the eyes of the Japanese when drawing the icon of “The Scrum Way” and was astounded!



Chapter 1: Introduction to Scrum

The Scrum Foundation provides an introduction to Scrum first developed at Yahoo and later updated by Pete Deemer and Gabrielle Benefield for their independent consulting practices. Tobias Meyer asked Jeff to help educate engineers at Yahoo on Scrum in November, 2005. Vice President of Development, Pete Deemer set up a Yahoo Senior Management meeting the following month for a Scrum briefing at an evening dinner in Palo Alto. By the end of the dinner, Yahoo management decided to roll out Scrum companywide. They felt Scrum fit the Yahoo style of development used in their early years as a startup company while giving them a structure that would support a global organization.

Pete recruited Gabrielle Benefield in 2006 to lead the Scrum rollout at Yahoo in the U.S. and then moved to India to become Yahoo Chief Product Officer and train the Indian teams in Scrum. Together, they have written their introduction to Scrum and provided some interesting survey data. In 2007, Gabrielle published updated survey data from almost 200 Scrum teams at Yahoo. These data show overwhelming support for Scrum as a "better, faster, cooler" method for building software. It also shows an annual Return on Investment (ROI) of 1000% for investment in Scrum trainers. This was the estimated ROI on Scrum training proposed to Pete Deemer in 2005 by Jeff Sutherland.

Jeff, Pete, and Gabrielle, along with Jens Ostergaard from Denmark joined forces to create the Scrum Foundation (formerly Scrum Training Institute) in 2008. This allows them to provide global training, consulting, and coaching services to those companies implementing Scrum. Since then the Scrum Foundation has expanded to include as partners many of the leading Scrum companies, Scrum trainers and Scrum coaches worldwide.

Scrum Primer Version 1.2

Pete Deemer and Gabrielle Benefield, Scrum Foundation, 2010
Criag Larman, craiglarman.com
Bas Vodde, Odd-e.com

A note to readers: There are many concise descriptions of Scrum available online, and this primer aims to provide the next level of detail on the practices. It is not intended as the final step in a Scrum education; teams that are considering adopting Scrum are advised to equip themselves with Ken Schwaber's *Agile Project Management with Scrum* or *Agile Software Development with Scrum*, and take advantage of the many excellent Scrum training and coaching options that are available; full details are at scrumalliance.org. Our thanks go to Ken Schwaber, Dr. Jeff Sutherland, and Mike Cohn for their generous input.

Traditional Software Development

The traditional way to build software, used by companies big and small, was a sequential life cycle commonly known as "the waterfall." There are many variants (such as the V-Model), but it typically begins with a detailed planning phase, where the end product is carefully thought through, designed, and documented in great detail. The tasks necessary to execute the design are determined, and the work is organized using tools such as Gantt charts and applications such as Microsoft Project. The team arrives at an estimate of how long the development will take by adding up detailed estimates of the individual steps involved. Once stakeholders have thoroughly reviewed the plan and provided their approvals, the team starts to work. Team members complete their specialized portion of the work, and then hand it off to others in production-line fashion. Once the work is complete, it is delivered to a testing organization (some call this Quality Assurance), which completes testing prior to the product reaching the customer. Throughout the process, strict controls are placed on deviations from the plan to ensure that what is produced is actually what was designed.

This approach has strengths and weaknesses. Its great strength is that it is supremely logical – think before you build, write it all down, follow a plan, and keep everything as organized as possible. It has just one great weakness: humans are involved.

For example, this approach requires that the good ideas all come at the beginning of the release cycle, where they can be incorporated into the plan. But as we all know, good ideas appear throughout the process – in the beginning, the middle, and sometimes even the day before launch, and a process that does not permit change will stifle this innovation. With the waterfall, a great idea late in the release cycle is not a gift, it's a threat.

The waterfall approach also places a great emphasis on writing things down as a primary method for communicating critical information. The very reasonable assumption is that if I can write down on paper as much as possible of what's in my head, it will more reliably make it into the head of everyone else on the team; plus, if it's on paper, there is tangible proof that I've done my job. The reality, though, is that most of the time these highly detailed 50-page requirements documents just do not get read. When they *do* get read, the misunderstandings are often compounded. A written document is an incomplete picture of my ideas; when you read it, you create another abstraction, which is now two steps away from what I *think* I meant to say at that time. It is no surprise that serious misunderstandings occur.

Something else that happens when you have humans involved is the hands-on "aha" moment – the first time that you actually use the working product. You immediately think of 20 ways you could have made it better. Unfortunately, these very valuable insights often come at the end of the release cycle, when changes are most difficult and disruptive – in other words, when doing the right thing is most expensive, at least when using a traditional method.

Humans are not able to predict the future. For example, your competition makes an announcement that was not expected. Unanticipated technical problems crop up that force a change in direction. Furthermore, people are particularly bad at planning uncertain things far into

the future – guessing today how you will be spending your week eight months from now is something of a fantasy. It has been the downfall of many a carefully constructed Gantt chart. In addition, a sequential life cycle tends to foster an adversarial relationship between the people that are handing work off from one to the next. “He’s asking me to build something that’s not in the specification.” “She’s changing her mind.” “I can’t be held responsible for something I don’t control.” And this gets us to another observation about sequential development – it is not much fun. The waterfall model is a cause of great misery for the people who build products. The resulting products fall well short of expressing the creativity, skill, and passion of their creators. People are not robots, and a process that requires them to act like robots results in unhappiness. A rigid, change-resistant process produces mediocre products. Customers may get what they first ask for (at least two translation steps removed), but is it what they really want once they see the product? By gathering all the requirements up front and having them set in stone, the product is condemned to be only as good as the initial idea, instead of being the best once people have learned or discovered new things.

Many practitioners of a sequential life cycle experience these shortcomings again and again. But, it seems so supremely logical that the common reaction is to turn inward: “If only we did it better, it would work” – if we just planned more, documented more, resisted change more, everything would work smoothly. Unfortunately, many teams find just the opposite: the harder they try, the worse it gets! There are also management teams that have invested their reputation – and many resources – in a waterfall model; changing to a fundamentally different model is an apparent admission of having made a mistake. And Scrum *is* fundamentally different...

Agile Development and Scrum

The agile family of development methods were born out of a belief that an approach more grounded in human reality – and the product development reality of learning, innovation, and change – would yield better results. Agile principles emphasize building working software that people can get hands on quickly, versus spending a lot of time writing specifications up front. Agile development focuses on cross-functional teams empowered to make decisions, versus big hierarchies and compartmentalization by function. And it focuses on rapid iteration, with continuous customer input along the way. Often when people learn about agile development or Scrum, there’s a glimmer of recognition – it sounds a lot like back in the start-up days, when we “just did it.”

By far the most popular agile method is Scrum. It was strongly influenced by a 1986 *Harvard Business Review* article on the practices associated with successful product development groups; in this paper the terms “Rugby” and “Scrum” were introduced, which later morphed into “Scrum” in *Wicked Problems, Righteous Solutions* (1991, DeGrace and Stahl) relating successful development to the game of Rugby in which a self-organizing team moves *together* down the field of product development. It was then formalized in 1995 by Ken Schwaber and Dr. Jeff Sutherland. Scrum is now used by companies large and small, including Yahoo!, Microsoft, Google, Lockheed Martin, Motorola, SAP, Cisco, GE, CapitalOne and the US Federal Reserve. Many teams using Scrum report significant improvements, and in some cases complete transformations, in both productivity and morale. For product developers – many of whom have been burned by the “management fad of the month club” – this is significant. Scrum is simple and powerful.

Scrum Summary

Scrum is an iterative, incremental framework for projects and product or application development. It structures development in cycles of work called **Sprints**. These iterations are no more than one month each, and take place one after the other without pause. The Sprints are *timeboxed* – they end on a specific date whether the work has been completed or not, and are *never extended*. At the beginning of each Sprint, a cross-functional team selects **items** (customer requirements) from a prioritized list. The team commits to complete the items by the end of the Sprint. During the Sprint, the chosen items do not change. Every day the team gathers briefly to inspect its progress, and adjust the next steps needed to complete the work remaining. At the end of the Sprint, the team reviews the Sprint with stakeholders, and demonstrates what it has built. People obtain feedback that can be incorporated in the next Sprint. Scrum emphasizes working product at the end of the Sprint that is really "done"; in the case of software, this means code that is integrated, fully tested and potentially shippable. Key roles, artifacts, and events are summarized in Figure 1.

A major theme in Scrum is "inspect and adapt." Since development inevitably involves learning, innovation, and surprises, Scrum emphasizes taking a short step of development, inspecting both the resulting product and the efficacy of current practices, and then adapting the product goals and process practices. Repeat forever.

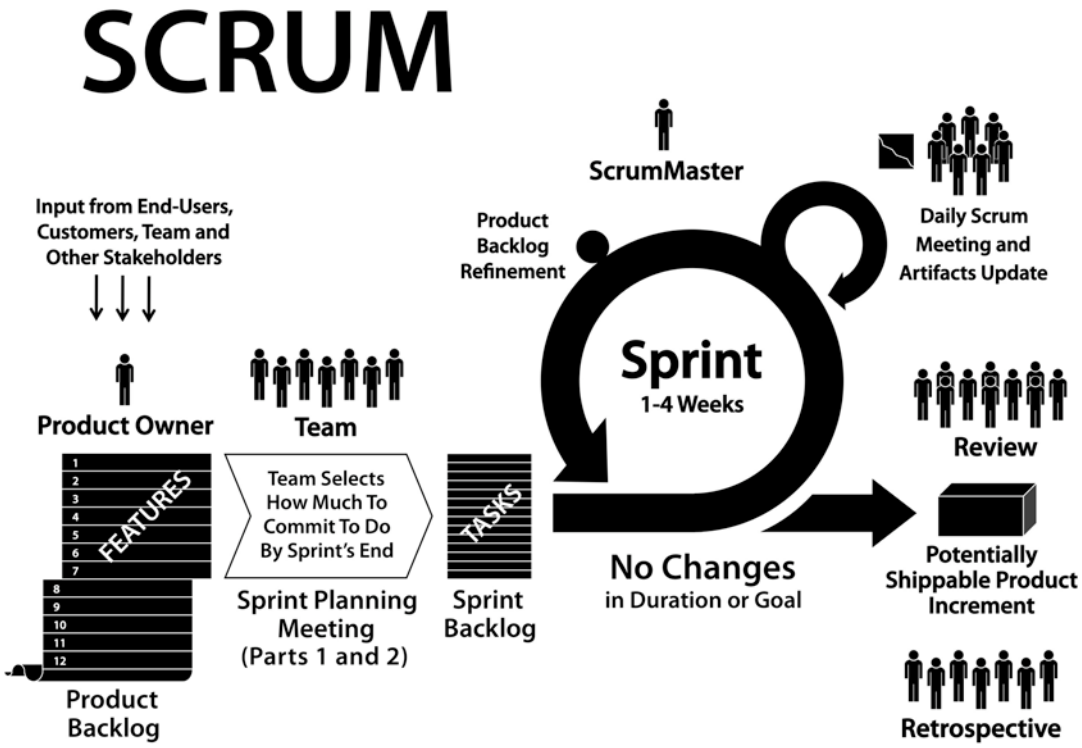


Figure 1. Scrum

Scrum Roles

In Scrum, there are three roles: The Product Owner, The Team, and The ScrumMaster. Together these are known as The Scrum Team. The **Product Owner** is responsible for maximizing return on investment (ROI) by identifying product features, translating these into a prioritized list, deciding which should be at the top of the list for the next Sprint, and continually re-prioritizing and refining the list. The Product Owner has profit and loss responsibility for the product, assuming it is a commercial product. In the case of an internal application, the Product Owner is not responsible for ROI in the sense of a commercial product (that will generate revenue), but they are still responsible for maximizing ROI in the sense of choosing – each Sprint – the highest-business-value lowest-cost items. In practice, 'value' is a fuzzy term and prioritization may be influenced by the desire to satisfy key customers, alignment with strategic objectives, attacking risks, improving, and other factors. In some cases, the Product Owner and the customer are the same person; this is common for internal applications. In others, the customer might be millions of people with a variety of needs, in which case the Product Owner role is similar to the Product Manager or Product Marketing Manager position in many product organizations. However, the Product Owner is somewhat different than a traditional Product Manager because they actively and frequently interact with the Team, personally offering the priorities and reviewing the results each two- or four-week iteration, rather than delegating development decisions to a project manager. It is important to note that in Scrum there is one and only one person who serves as – and has the final authority of – Product Owner, and he or she is responsible for the value of the work.

The Team builds the product that the Product Owner indicates: the application or website, for example. The Team in Scrum is "cross-functional" – it includes all the expertise necessary to deliver the potentially shippable product each Sprint – and it is "self-organizing" (self-managing), with a very high degree of autonomy and accountability. The Team decides what to commit to, and how best to accomplish that commitment; in Scrum lore, the Team is known as "Pigs" and everyone else in the organization are "Chickens" (which comes from a joke about a pig and a chicken deciding to open a restaurant called "Ham and Eggs," and the pig having second thoughts because "he would be truly committed, but the chicken would only be involved").

The Team in Scrum is seven plus or minus two people, and for a software product the Team might include people with skills in analysis, development, testing, interface design, database design, architecture, documentation, and so on. The Team develops the product and provides ideas to the Product Owner about how to make the product great. In Scrum the Teams are most productive and effective if all members are 100 percent dedicated to the work for one product during the Sprint; avoid multitasking across multiple products or projects. Stable teams are associated with higher productivity, so avoid changing Team members. Application groups with many people are organized into multiple Scrum Teams, each focused on different features for the product, with close coordination of their efforts. Since one team often does all the work (planning, analysis, programming, and testing) for a complete customer-centric feature, Teams are also known as *feature teams*.

The **ScrumMaster** helps the product group learn and apply Scrum to achieve business value. The ScrumMaster does whatever is in their power to help the Team and Product Owner be

successful. The ScrumMaster is *not* the manager of the Team or a project manager; instead, the ScrumMaster serves the Team, protects them from outside interference, and educates and guides the Product Owner and the Team in the skillful use of Scrum. The ScrumMaster makes sure everyone (including the Product Owner, and those in management) understands and follows the practices of Scrum, and they help lead the organization through the often difficult change required to achieve success with agile development. Since Scrum makes visible many impediments and threats to the Team's and Product Owner's effectiveness, it is important to have an engaged ScrumMaster working energetically to help resolve those issues, or the Team or Product Owner will find it difficult to succeed. There should be a dedicated full-time ScrumMaster, although a smaller Team might have a team member play this role (carrying a lighter load of regular work when they do so). Great ScrumMasters can come from any background or discipline: Engineering, Design, Testing, Product Management, Project Management, or Quality Management.

The ScrumMaster and the Product Owner cannot be the same individual; at times, the ScrumMaster may be called upon to push back on the Product Owner (for example, if they try to introduce new deliverables in the middle of a Sprint). And unlike a project manager, the ScrumMaster does not tell people what to do or assign tasks – they facilitate the process, supporting the Team as it organizes and manages itself. If the ScrumMaster was previously in a position managing the Team, they will need to significantly change their mindset and style of interaction for the Team to be successful with Scrum.

Note there is no role of project manager in Scrum. This is because none is needed; the traditional responsibilities of a project manager have been divided up and reassigned among the three Scrum roles. Sometimes an (ex-)project manager can step into the role of ScrumMaster, but this has a mixed record of success – there is a fundamental difference between the two roles, both in day-to-day responsibilities and in the mindset required to be successful. A good way to understand thoroughly the role of the ScrumMaster, and start to develop the core skills needed for success, is the Scrum Alliance's Certified ScrumMaster training.

In addition to these three roles, there are other contributors to the success of the product, including functional managers (for example, an engineering manager). While their role changes in Scrum, they remain valuable. For example:

- they support the Team by respecting the rules and spirit of Scrum
- they help remove impediments that the Team and Product Owner identify
- they make their expertise and experience available

In Scrum, these individuals replace the time they previously spent playing the role of "nanny" (assigning tasks, getting status reports, and other forms of micromanagement) with time as "guru" and "servant" of the Team (mentoring, coaching, helping remove obstacles, helping problem-solve, providing creative input, and guiding the skills development of Team members). In this shift, managers may need to change their management style; for example, using Socratic questioning to help the Team discover the solution to a problem, rather than simply deciding a solution and assigning it to the Team.

Starting Scrum

The first step in Scrum is for the Product Owner to articulate the product vision. Eventually, this evolves into a refined and prioritized list of features called the **Product Backlog**. This backlog exists (and evolves) over the lifetime of the product; it is the product road map (**Figure 2**). At any point, the Product Backlog is the single, definitive view of "everything that could be done by the Team ever, in order of priority". Only a single Product Backlog exists; this means the Product Owner is required to make prioritization decisions across the entire spectrum, representing the interest of stakeholders and influenced by the team.

					New Estimates of Effort					
					Remaining as of Sprint...					
Item	Details (wiki URL)	Priority	Estimate of Value	Initial Estimate of Effort	1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart (see UI sketches on wiki page)	...	1	7	5						
As a buyer, I want to remove a book in a shopping cart	...	2	6	2						
Improve transaction processing performance (see target performance metrics on wiki)	...	3	6	13						
Investigate solutions for speeding up credit card validation (see target performance metrics on wiki)	...	4	6	20						
Upgrade all servers to Apache 2.2.3	...	5	5	13						
Diagnose and fix the order processing script errors (bugzilla ID 14823)	...	6	2	3						
As a shopper, I want to create and save a wish list	...	7	7	40						
As a shopper, I want to to add or delete items on my wish list	...	8	4	20						

Figure 2. The Product Backlog

The Product Backlog includes a variety of **items**, primarily new customer features ("enable all users to place book in shopping cart"), but also engineering improvement goals ("rework the transaction processing module to make it scalable"), exploratory or research work ("investigate solutions for speeding up credit card validation"), and, possibly, known defects ("diagnose and fix the order processing script errors"), if there are only a few problems. (A system with many defects usually has a separate defect tracking system.) The Product Backlog can be articulated in any way that is clear and sustainable, though either Use Cases or "user stories" are often used to describe the Product Backlog items in terms of their value to the end user of the product.

The subset of the Product Backlog that is intended for the current release is known as the **Release Backlog**, and in general, this portion is the primary focus of the Product Owner. The Product Backlog is continuously updated by the Product Owner to reflect changes in the needs of the customer, new ideas or insights, moves by the competition, technical hurdles that appear, and so forth. The Team provides the Product Owner with estimates of the effort required for each item on the Product Backlog. In addition, the Product Owner is responsible for assigning a *business value estimate* to each individual item. This is usually an unfamiliar practice for a Product Owner. As such, it is something a ScrumMaster may help the Product Owner learn to do. With these two estimates (effort and value) and perhaps with additional risk estimates, the Product Owner prioritizes the backlog (actually, usually just the Release Backlog subset) to maximize ROI (choosing items of high value with low effort) or secondarily, to reduce some major risk. As will be seen, these effort and value estimates may be refreshed each Sprint as

people learn; consequently, this is a continuous re-prioritization activity as the Product Backlog is ever-evolving.

Scrum does not define techniques for expressing or prioritizing items in the Product Backlog and it does not define an estimation technique. A common technique is to estimate in terms of relative size (factoring in effort, complexity, and uncertainty) using a unit of "story points" or simply "points".

Over time, a Team tracks how much work it can do each Sprint; for example, averaging 26 points per Sprint. With this information they can project a release date to complete all features, or how many features can be completed by a fixed date, if the average continues and nothing changes. This average is called the "velocity" of the team. Velocity is expressed in the same units as the Product Backlog item size estimates.

The items in the Product Backlog can vary significantly in size or effort. Larger ones are broken into smaller items during the Product Backlog Refinement workshop or the Sprint Planning Meeting, and smaller ones may be consolidated. The Product Backlog items for the upcoming next several Sprints should be small and fine-grained enough that they are understood by the Team, enabling commitments made in the Sprint Planning meeting to be meaningful; this is called an "actionable" size.

One of the myths about Scrum is that it prevents you from writing detailed specifications; in reality, it is up to the Product Owner and Team to decide how much detail is required, and this will vary from one backlog item to the next, depending on the insight of the Team, and other factors. State what is important in the least amount of space necessary – in other words, do not describe every possible detail of an item, just make clear what is necessary for it to be understood. Low priority items, far from being implemented and usually "coarse grained" or large, have less requirements details. High priority and fine-grained items that will soon be implemented tend to have more detail.

Sprint Planning

At the beginning of each Sprint, the **Sprint Planning Meeting** takes place. It is divided into two distinct sub-meetings, the first of which is called **Sprint Planning Part One**.

In Sprint Planning Part One, the Product Owner and Team (with facilitation from the ScrumMaster) review the high-priority items in the Product Backlog that the Product Owner is interested in implementing this Sprint. They discuss the goals and context for these high-priority items on the Product Backlog, providing the Team with insight into the Product Owner's thinking. The Product Owner and Team also review the "Definition of Done" (which was established earlier) that all items must meet, such as, "Done means coded to standards, reviewed, implemented with unit test-driven development (TDD), tested with 100 percent test automation, integrated, and documented." Part One focuses on understanding *what* the Product Owner wants. According to the rules of Scrum, at the end of Part One the (always busy) Product Owner may leave although they *must* be available (for example, by phone) during the next meeting. However, they are welcome to attend Part Two...

Sprint Planning Part Two focuses on detailed task planning for *how* to implement the items that the Team decides to take on. The Team selects the items from the Product Backlog they commit to complete by the end of the Sprint, starting at the top of the Product Backlog (in others words, starting with the items that are the highest priority for the Product Owner) and working down the list in order. This is a key practice in Scrum: The Team decides how much work it will commit to complete, rather than having it assigned to them by the Product Owner. This makes for a more reliable commitment because the Team is making it based on its own analysis and planning, rather than having it decided by someone else. While the Product Owner does not have control over how much the Team commits to, he or she knows that the items the Team is committing to are drawn from the top of the Product Backlog – in other words, the items that he or she has rated as most important. The Team has the ability to lobby for items from further down the list; this usually happens when the Team and Product Owner realize that something of lower priority fits easily and appropriately with the high priority items.

The Sprint Planning Meeting will often last a number of hours, but no more than eight hours for a four-week Sprint – the Team is making a serious commitment to complete the work, and this commitment requires careful thought to be successful. The Team will probably begin the Sprint Planning Part Two by estimating how much time each member has for Sprint-related work – in other words, their average workday minus the time they spend attending meetings, doing email, taking lunch breaks, and so on. For most people this works out to 4-6 hours of time per day available for Sprint-related work. This is the team's capacity for the upcoming Sprint. See Figure 3.

Sprint Length	2 weeks
Workdays during Sprint	8 days

Team Member	Available Days During Sprint*	Available Hours per Day	Total Available Hours
Tracy	8	4	32
Sanjay	7	5	35
Phillip	8	4	32
Jing	6	5	30

* Net of vacation and other days out of office

Figure 3. Estimating Available Hours

Once the capacity is determined, the Team figures out how many Product Backlog items they can complete in that time, and how they will go about completing them. This often starts with a design discussion at a whiteboard. Once the overall design is understood, the Team decomposes the Product Backlog items into work. The Team starts with the first item on the Product Backlog – in other words, the Product Owner's highest priority item – and working together, breaks it down into individual tasks, which are recorded in a document called the **Sprint Backlog** (Figure 4).

As mentioned, the Product Owner must be available during Part Two (such as via the phone) so that clarification is possible. The Team will move sequentially down the Product Backlog in this

way, until it's used up all its estimated capacity. At the end of the meeting, the Team will have produced a list of all the tasks with estimates (typically in hours or fractions of a day).

Scrum encourages multi-skilled workers, rather than only "working to job title" such as a "tester" only doing testing. In other words, Team members "go to where the work is" and help out as possible. If there are many testing tasks, then *all* Team members may help. This does not imply that everyone is a generalist; no doubt some people are especially skilled in testing (and so on) but Team members work together and learn new skills from each other. Consequently, during task generation and estimation in Sprint Planning, it is not necessary – nor appropriate – for people to volunteer for all the tasks "they can do best." Rather, it is better to only volunteer for one task at a time, when it is time to pick up a new task, and to consider choosing tasks that will on purpose involve learning (perhaps by pair work with a specialist).

All that said, there are *rare* times when *John* may do a particular task because it would take far too long or be impossible for others to learn – perhaps John is the only person with any artistic skill to draw pictures. Other Team members could not draw a "stick man" if their life depended on it. In this rare case – and if it is not rare and not getting rarer as the Team learns, there is something wrong – it may be necessary to ask if the total planned drawing tasks that *must* be done by John are feasible within the short Sprint.

Many Teams also make use of a visual task-tracking tool, in the form of a wall-sized task board where tasks (written on Post-It Notes) migrate during the Sprint across columns labeled "Not Yet Started," "In Progress," and "Completed." See Figure 5.

				New Estimates of Effort Remaining as of Day...					
Product Backlog Item	**Sprint Task**	**Volunteer**	**Initial Estimate of Effort**	**1**	**2**	**3**	**4**	**5**	**6**
As a buyer, I want to place a book in a shopping cart	modify database		5						
	create webpage (UI)		8						
	create webpage (Javascript logic)		13						
	write automated acceptance tests		13						
	update buyer help webpage		3						
	. . .								
Improve transaction processing performance	merge DCP code and complete layer-level tests		5						
	complete machine order for pRank		8						
	change DCP and reader to use pRank http API		13						

Figure 4. Sprint Backlog

Figure 5. Visual Management - Sprint Backlog tasks on the wall

One of the pillars of Scrum is that once the Team makes its commitment, any additions or changes must be deferred until the next Sprint. This means that if halfway through the Sprint the Product Owner decides there is a new item he or she would like the Team to work on, he cannot make the change until the start of the next Sprint. If an external circumstance appears that significantly changes priorities, and means the Team would be wasting its time if it continued working, the Product Owner or the Team can terminate the Sprint. The Team stops, and a new Sprint Planning meeting initiates a new Sprint. The disruption of doing this is usually great; this serves as a disincentive for the Product Owner or Team to resort to this dramatic decision.
There is a powerful, positive influence that comes from the Team being protected from changing goals during the Sprint. First, the Team gets to work knowing with absolute certainty that its commitments will not change, that reinforces the Team's focus on ensuring completion. Second, it disciplines the Product Owner into really thinking through the items he or she prioritizes on the Product Backlog and offers to the Team for the Sprint.
By following these Scrum rules the Product Owner gains two things. First, he or she has the confidence of knowing the Team has made a commitment to complete a realistic and clear set of work it has chosen. Over time a Team can become quite skilled at choosing and delivering on a realistic commitment. Second, the Product Owner gets to make whatever changes he or she likes to the Product Backlog before the start of the *next* Sprint. At that point, additions, deletions, modifications, and re-prioritizations are all possible and acceptable. While the Product Owner is not able to make changes to the selected items under development during the current Sprint, he or she is only one Sprint's duration or less away from making any changes they wish. Gone is the stigma around change – change of direction, change of requirements, or just plain changing your mind – and it may be for this reason that Product Owners are usually as enthusiastic about Scrum as anyone.

Daily Scrum

Once the Sprint has started, the Team engages in another of the key Scrum practices: The **Daily Scrum**. This is a short (15 minutes or less) meeting that happens every workday at an appointed time. Everyone on the Team attends. To keep it brief, it is recommended that everyone remain standing. It is the Team's opportunity to synchronize their work and report to each other on obstacles. In the Daily Scrum, one by one, each member of the Team reports three (and only three) things *to the other members of the Team*: (1) What they were able to get done since the

last meeting; (2) what they are planning to finish by the next meeting; and (3) any blocks or impediments that are in their way.

Note that the Daily Scrum is not a status meeting to report to a manager; it is a time for a self-organizing Team to share with each other what is going on, to help them coordinate. Someone makes note of the blocks, and the ScrumMaster is responsible to help Team members resolve them. There is no discussion during the Daily Scrum, only reporting answers to the three questions; if discussion is required it takes place immediately after the Daily Scrum in a follow-up meeting, although in Scrum no one is required to attend this. This follow-up meeting is a common event where the Team adapts to the information they heard in the Daily Scrum: in other words, another inspect and adapt cycle. It is generally recommended *not* to have managers or others in positions of perceived authority attend the Daily Scrum. This risks making the Team feel "monitored" – under pressure to report major progress every day (an unrealistic expectation), and inhibited about reporting problems – and it tends to undermine the Team's self-management, and invite micromanagement. It would be more useful for a stakeholder to instead reach out to the Team following the meeting, and offer to help with any blocks that are slowing the Team's progress.

Updating Sprint Backlog & Sprint Burndown Chart

The Team in Scrum is self-managing, and in order to do this successfully, it must know how it is doing. Every day, the Team members update their estimate of the amount of time remaining to complete their current task in the **Sprint Backlog** (Figure 6). Following this update, someone adds up the hours remaining for the Team as a whole, and plots it on the **Sprint Burndown Chart** (Figure 7). This graph shows, each day, a new estimate of how much work (measured in person hours) remains until the Team's tasks are finished. Ideally, this is a *downward* sloping graph that is on a trajectory to reach "zero effort remaining" by the last day of the Sprint. Hence it is called a *burndown* chart. And while sometimes it looks good, often it does not; this is the reality of product development. The important thing is that it shows the Team their progress towards their goal, not in terms of how much time was *spent* in the past (an irrelevant fact in terms of *progress*), but in terms of how much work *remains in the future* – what separates the Team from their goal. If the burndown line is not tracking downwards towards completion near the end of the Sprint, then the Team needs to adjust, such as to reduce the scope of the work or to find a way to work more efficiently while still maintaining a sustainable pace.

While the Sprint Burndown chart can be created and displayed using a spreadsheet, many Teams find it is more effective to show it on paper on a wall in their workspace, with updates in pen; this "low-tech/high-touch" solution is fast, simple, and often more visible than a computer chart.

				New Estimates of Effort Remaining at end of Day...					
Product Backlog Item	Sprint Task	Volunteer	Initial Estimate of Effort	1	2	3	4	5	6
As a buyer, I want to place a book in a shopping cart	modify database	Sanjay	5	4	3	0	0	0	
	create webpage (UI)	Jing	3	3	3	2	0	0	
	create webpage (Javascript logic)	Tracy & Sam	2	2	2	2	1	0	
	write automated acceptance tests	Sarah	5	5	5	5	5	0	
	update buyer help webpage	Sanjay & Jing	3	3	3	3	3	0	
	. . .								
Improve transaction processing performance	merge DCP code and complete layer-level tests		5	5	5	5	5	5	
	complete machine order for pRank		3	3	8	8	8	8	
	change DCP and reader to use pRank http API		5	5	5	5	5	5	
	. . .	. . .	. . .						
		Total (person hours)	50	49	48	44	43	34	

Figure 6. Daily Updates of Work Remaining on the Sprint Backlog

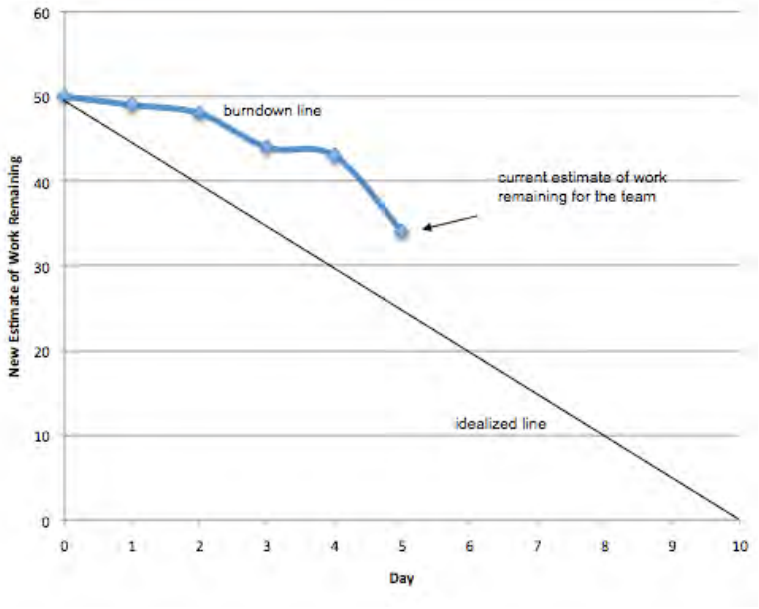


Figure 7. Sprint Burndown Chart

Product Backlog Refinement

One of the lesser known, but valuable, guidelines in Scrum is that five or ten percent of each Sprint must be dedicated by the Team to refining (or "grooming") the Product Backlog. This includes detailed requirements analysis, splitting large items into smaller ones, estimation of new items, and re-estimation of existing items. Scrum is silent on how this work is done, but a frequently used technique is a focused workshop near the end of the Sprint, so that the Team and Product Owner can dedicate themselves to this work without interruption.

For a two-week Sprint, five percent of the duration implies that each Sprint there is a half-day Product Backlog Refinement workshop. This refinement activity is not for items selected for the current Sprint; it is for items for the future, most likely in the next one or two Sprints. With this practice, Sprint Planning becomes relatively simple because the Product Owner and Scrum Team start the planning with a clear, well-analyzed and carefully estimated set of items. A sign that this refinement workshop is not being done (or not being done well) is that Sprint Planning involves

significant questions, discovery, or confusion and feels incomplete; planning work then often spills over into the Sprint itself, which is typically not desirable.

Ending the Sprint

One of the core tenets of Scrum is that the duration of the Sprint is never extended – it ends on the assigned date regardless of whether the Team has completed the work it committed to. A Team typically over-commits in its first few Sprints and fails to accomplish its commitments. Sometimes it then overcompensates and under-commits, and finishes early (in which case it can ask the Product Owner for more Product Backlog items to work on). But by the third or fourth Sprint a Team has typically figured out what it is capable of delivering (most of the time), and they will meet their Sprint goals more reliably after that. Teams are encouraged to pick one duration for their Sprints (say, two weeks) and not change it. This helps the Team learn how much it can accomplish, which helps in both estimation and longer-term release planning. It also helps the Team achieve a rhythm for their work; this is often referred to as the "heartbeat" of the Team in Scrum.

Sprint Review

After the Sprint ends, there is the **Sprint Review**, where the Team and the Product Owner review the Sprint. This is often mislabeled the "demo" but that does not capture the real intent of this meeting. A key idea in Scrum is *inspect and adapt*. To see and learn what is going on and then evolve based on feedback, in repeating cycles. The Sprint Review is an inspect and adapt activity for the *product*. It is a time for the Product Owner to learn what is going on with the product and with the Team (that is, a review of the Sprint); and for the Team to learn what is going on with the Product Owner and the market. Consequently, the most important element of the Review is an in-depth *conversation* between the Team and Product Owner to learn the situation, to get advice, and so forth. The review includes a demo of what the Team built during the Sprint, but if the focus of the review is a demo rather than conversation, there is an imbalance.

A useful – but often overlooked – Scrum guideline is that it the ScrumMaster's responsibility to ensure that everyone knows the "Definition of Done" defined for this product or release. He prevents the team from demonstrating or discussing Product Backlog Items that are not 'done' according to the "Definition of Done." Items that are not 'done' go back to the Product Backlog and will be re-prioritized by the Product Owner. In way, there is transparency regarding the quality of the work; Teams cannot fake the quality by presenting software that appears to work well, but may be implemented with a messy pile of poor quality and untested code.

Present at this meeting are the Product Owner, Team members, and ScrumMaster, plus customers, stakeholders, experts, executives, and anyone else interested. The demo portion of the Sprint Review is not a "presentation" the Team gives – there is no slideware. A guideline in Scrum is that no more than 30 minutes should be spent preparing for the review, otherwise it suggests something is wrong with the work of the Team. It is simply a demo of what has been built. Anyone present is free to ask questions and give input.

Sprint Retrospective

The Sprint Review involves inspect and adapt regarding the *product*. The **Sprint Retrospective**, which follows the Review, involves inspect and adapt regarding the *process*. This is a practice that some Teams skip, and that's unfortunate, because it's the main mechanism for taking the visibility that Scrum provides into areas of potential improvement, and turning it into results. It's an opportunity for the Team to discuss what's working and what's not working, and agree on changes to try. The Team and ScrumMaster will attend, and the Product Owner is welcome but not required to attend. Sometimes the ScrumMaster can act as an effective facilitator for the retrospective, but it may be better to find a neutral outsider to facilitate the meeting; a good approach is for ScrumMasters to facilitate each others' retrospectives, which enables cross-pollination among Teams.

There are many techniques for conducting a Sprint Retrospective, and the book *Agile Retrospectives* (Derby, Larsen 2006) provides a useful catalogue of techniques. A simple way to structure the discussion is to draw two columns on a whiteboard, labeled "What's Working Well" and "What Could Work Better" – and then go around the room, with each person adding one or more items to either list. As items are repeated, check marks are added next to them, so the common items become clear. Then the Team looks for underlying causes, and agrees on a small number of changes to try in the upcoming Sprint, along with a commitment to review the results at the next Sprint Retrospective.

A useful practice at the end of the Retrospective is for the Team to label each of the items in each column with either a "C" if it is *caused* by Scrum (in other words, without Scrum it would not be happening), or an "E" if it is *exposed* by Scrum (in other words, it would be happening with or without Scrum, but Scrum makes it known to the Team), or a "U" if it's unrelated to Scrum (like the weather). The Team may find a lot of C's on the "What's Working Well" side of the board, and a lot of E's on the "What Could Work Better "; this is good news, even if the "What Could Work Better" list is a long one, because the first step to solving underlying issues is making them visible, and Scrum is a powerful catalyst for that.

Updating Release Backlog & Burndown Chart

At this point, some items have been finished, some have been added, some have new estimates, and some have been dropped from the release goal. The Product Owner is responsible for ensuring that these changes are reflecting in the Release Backlog (and more broadly, the Product Backlog). In addition, Scrum includes a **Release Burndown** chart that shows progress towards the release date. It is analogous to the Sprint Burndown chart, but is at the higher level of items (requirements) rather than fine-grained tasks. Since a new Product Owner is unlikely to know why or how to create this chart, this is another opportunity for a ScrumMaster to help the Product Owner. See Figure 8 and Figure 9 for an example of the Release Backlog and Release Burndown chart.

					New Estimates of Effort Remaining at end of Sprint...					
Item	**Details (wiki URL)**	**Priority**	**Estimate of Value**	**Initial Estimate of Effort**	**1**	**2**	**3**	**4**	**5**	**6**
As a buyer, I want to place a book in a shopping cart (see UI sketches on wiki page)	...	1	7	5	0	0	0			
As a buyer, I want to remove a book in a shopping cart	...	2	6	2	0	0	0			
Improve transaction processing performance (see target performance metrics on wiki)	...	3	6	13	13	0	0			
Investigate solutions for speeding up credit card validation (see target performance metrics on wiki)	...	4	6	20	20	20	0			
Upgrade all servers to Apache 2.2.3	...	5	5	13	13	13	13			
Diagnose and fix the order processing script errors (bugzilla ID 14823)	...	6	2	3	3	3	3			
As a shopper, I want to create and save a wish list	...	7	7	40	40	40	40			
As a shopper, I want to to add or delete items on my wish list	...	8	4	20	20	20	20			
. . .				. . .	. . .					
			Total	**537**	**580**	**570**	**500**			

Figure 8. Release Backlog (a subset of the Product Backlog)

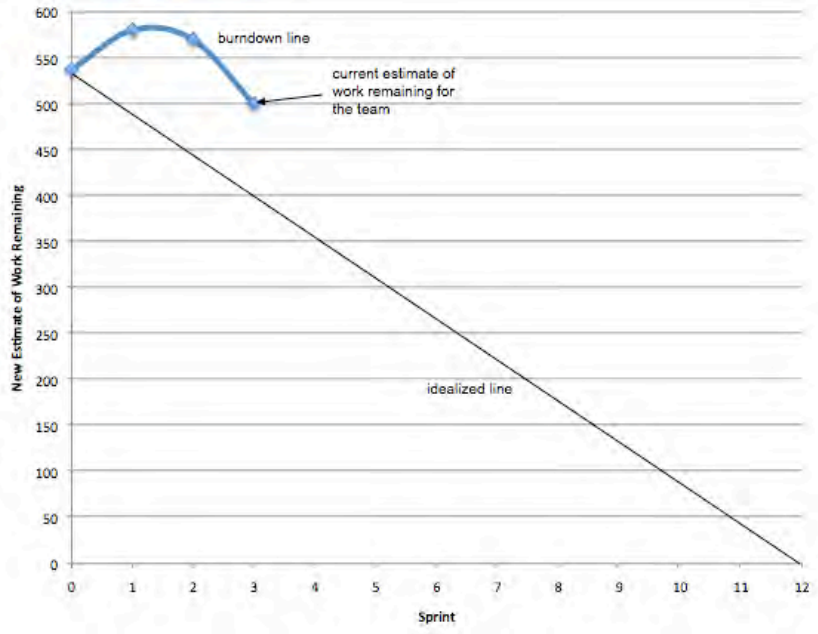


Figure 9. Release Burndown Chart

Starting the Next Sprint

Following the Sprint Review, the Product Owner may update the Product Backlog with any new insight. At this point, the Product Owner and Team are ready to begin another Sprint cycle. There is no down time between Sprints – Teams normally go from a Sprint Retrospective one afternoon into the next Sprint Planning the following morning (or after the weekend).
One of the principles of agile development is "sustainable pace", and only by working regular hours at a reasonable level can Teams continue this cycle indefinitely.

Release Sprint

The perfection vision of Scrum is that the product is potentially shippable at the end of each Sprint, which implies there is no wrap up work required, such as testing or documentation. The implication is that *everything* is completely *finished* every Sprint; that you could actually ship it or deploy it immediately after the Sprint Review. This means that each increment is a complete slice of the final product and gives complete transparency to the Product Owner and stakeholders. They know exactly where they are at the end of every Sprint.
However, many organizations have weak development practices, tools and infrastructure and cannot achieve this perfection vision, or there are other extenuating circumstances (such as, "the machine broke"). In this case, there will be some remaining work, such as final production environment integration testing, and so there will be the need for a "Release Sprint" to handle this remaining work.

Note that the need for a Release Sprint is a sign of some weakness; the ideal is that it is not required. When necessary, Sprints continue until the Product Owner decides the product is almost ready for release, at which point there will be a Release Sprint to prepare for launch. If the Team has followed good development practices, with continuous refactoring and integration, and effective testing during each Sprint, there should be little pre-release stabilization or other wrap-up work required.

Release Planning & Initial Product Backlog Refinement

A question that is sometimes asked is how, in an iterative model, can long-term release planning be done. There are two cases to consider: (1) a new product in its first release, and (2) an existing product in a later release.

In the case of a new product, or *an existing product just adopting Scrum*, there is the need to do initial Product Backlog refinement before the first Sprint, where the Product Owner and Team shape a proper Scrum Product Backlog. This could take a few days or a week, and involves a vision workshop, some detailed requirements analysis, and estimation of all the items identified for the first release.

Surprisingly in Scrum, in the case of an established product with an established Product Backlog, there should not be the need for any special or extensive release planning for the next release. Why? Because the Product Owner and Team should be doing Product Backlog refinement every Sprint (five or ten percent of each Sprint), continuously preparing for the future. This *continuous product development* mode obviates the need for the dramatic punctuated prepare-execute-conclude stages one sees in traditional sequential life cycle development.

During an initial Product Backlog refinement workshop and during the continuous backlog refinement each Sprint, the Team and Product Owner will do release planning, refining the estimates, priorities, and content as they learn.

Some releases are date-driven; for example: "We will release version 2.0 of our project at a trade-show on November 10." In this situation, the Team will complete as many Sprints (and build as many features) as is possible in the time available. Other products require certain

features to be built before they can be called complete and the product will not launch until these requirements are satisfied, however long that takes. Since Scrum emphasizes producing potentially shippable code each Sprint, the Product Owner may choose to start doing interim releases, to allow the customer to reap the benefits of completed work sooner.

Since they cannot possibly know everything up front, the focus is on creating and refining a plan to give the release broad direction, and clarify how tradeoff decisions will be made (scope versus schedule, for example). Think of this as the roadmap guiding you towards your final destination; which exact roads you take and the decisions you make during the journey may be determined en route.

Most Product Owners choose one release approach. For example, they will decide a release date, and will work with the Team to estimate the Release Backlog items that can be completed by that date. In situations where a "fixed price / fixed date / fixed deliverable" commitment is required – for example, contract development – one or more of those parameters must have a built-in buffer to allow for uncertainty and change; in this respect, Scrum is no different from other approaches.

Application or Product Focus

For applications or products – either for the market or for internal use within an organization – Scrum moves groups away from the older *project*-centric model toward a *continuous application/product development* model. There is no longer a project with a beginning, middle, and end. And hence no traditional project manager. Rather, there is simply a stable Product Owner and a long-lived self-managing Team that collaborate in an "endless" series of fixed-length Sprints, until the product or application is retired. All necessary "project" management work is handled by the Team and the Product Owner – who is an internal business customer or from Product Management. It is not managed by an IT manager or someone from a Project Management Office.

Scrum can also be used for true *projects* that are one-time initiatives (rather than work to create or evolve long-lived applications); still, in this case the Team and Product Owner do the project management.

What if there is insufficient new work from one or more existing applications to warrant a dedicated long-lived Team for each application? In this case, a stable long-lived Team may take on items from one application in one Sprint, and then items from another in the next Sprint; in this situation the Sprints are often quite short, such as one week.

Occasionally, there is insufficient new work even for the prior solution, and the Team may take on items from *several* applications during the same Sprint; however, beware this solution as it may devolve into unproductive multitasking across multiple applications. A basic productivity theme in Scrum is for the Team to be *focused* on one product or application for one Sprint.

Common Challenges

Scrum is not only a concrete set of practices – rather, and more importantly, it is a framework that provides transparency, and a mechanism that allows "inspect and adapt". Scrum works by making visible the dysfunction and impediments that are impacting the Product Owner and the Team's effectiveness, so that they can be addressed. For example, the Product Owner may not really know the market, the features, or how to estimate their relative business value. Or the Team may be unskillful in effort estimation or development work.

The Scrum framework will quickly reveal these weaknesses. Scrum does not solve the problems of development; it makes them painfully visible, and provides a framework for people to explore ways to resolve problems in short cycles and with small improvement experiments.

Suppose the Team fails to deliver what they committed to in the first Sprint due to poor task analysis and estimation skill. To the Team, this feels like failure. But in reality, this experience is the necessary first step toward becoming more realistic and thoughtful about its commitments. This pattern – of Scrum helping make visible dysfunction, enabling the Team to do something about it – is the basic mechanism that produces the most significant benefits that Teams using Scrum experience.

One common mistake made, when presented with a Scrum practice that is challenging, is to change Scrum. For example, Teams that have trouble delivering on their Sprint commitment might decide to make the Sprint duration extendable, so it never runs out of time – and in the process, ensure it never has to learn how to do a better job of estimating and managing its time. In this way, without coaching and the support of an experienced ScrumMaster, organizations can mutate Scrum into just a mirror image of its own weaknesses and dysfunction, and undermine the real benefit that Scrum offers: Making visible the good and the bad, and giving the organization the choice of elevating itself to a higher level.

Another common mistake is to assume that a practice is discouraged or prohibited just because Scrum does not specifically require it. For example, Scrum does not require the Product Owner to set a long-term strategy for his or her product; nor does it require engineers to seek advice from more experienced engineers about complex technical problems. Scrum leaves it to the individuals involved to make the right decision; and in most cases, both of these practices (along with many others) are well advised.

Something else to be wary of is managers imposing Scrum on their Teams; Scrum is about giving a Team space and tools to manage itself, and having this dictated from above is not a recipe for success. A better approach might begin with a Team learning about Scrum from a peer or manager, getting comprehensively educated in professional training, and then making a decision as a Team to follow the practices faithfully for a defined period; at the end of that period, the Team will evaluate its experience, and decide whether to continue.

The good news is that while the first Sprint is usually very challenging to the Team, the benefits of Scrum tend to be visible by the end of it, leading many new Scrum Teams to exclaim: "Scrum is hard, but it sure is a whole lot better than what we were doing before!"

Appendix: Terminology

Burn Down
The trend of work remaining across time in a Sprint, a Release, or a Product. The source of the raw data is the Sprint Backlog and the Product Backlog, with work remaining tracked on the vertical axis and the time periods (days of a Sprint, or Sprints) tracked on the horizontal axis.

Chicken
Someone who is interested in the project but does not have formal Scrum responsibilities and accountabilities (Team, Product Owner, ScrumMaster).

Daily Scrum
A short meeting held daily by each Team during which the Team members inspect their work, synchronize their work and progress and report and impediments to the ScrumMaster for removal. Follow-on meetings to adapt upcoming work to optimize the Sprint may occur after the Daily Scrum meetings.

Done
Complete as mutually agreed to by all parties and that conforms to an organization's standards, conventions, and guidelines. When something is reported as "done" at the Sprint Review meeting, it must conform to this agreed definition.

Estimated Work Remaining (Sprint Backlog items)
The number of hours that a Team member estimates remain to be worked on any task. This estimate is updated at the end of every day when the Sprint Backlog task is worked on. The estimate is the total estimated hours remaining, regardless of the number of people that perform the work.

Increment
Product functionality that is developed by the Team during each Sprint that is potentially shippable or of use to the Product Owner's stakeholders.

Increment of Potentially Shippable Product Functionality
A complete slice of the overall product or system that could be used by the Product Owner or stakeholders if they chose to implement it.

Sprint
An iteration, or one repeating cycle of similar work, that produces increment of product or system. No longer than one month and usually more than one week. The duration is fixed throughout the overall work and all teams working on the same system or product use the same length cycle.

Pig
Someone exercising one of the three Scrum roles (Team, Product Owner, ScrumMaster) who has made a commitment and has the authority to fulfill it.

Product Backlog
A prioritized list of requirements with estimated times to turn them into completed product functionality. Estimates are more precise the higher an item is in the Product Backlog priority.. The list emerges, changing as business conditions or technology changes.

Product Backlog Item
Functional requirements, non-functional requirements, and issues, prioritized in order of importance to the business and dependencies and estimated. The precision of the estimate depends on the priority and granularity of the Product Backlog item, with the highest priority items that may be selected in the next Sprint being very granular and precise.

Product Owner
The person responsible for managing the Product Backlog so as to maximize the value of the project. The Product Owner is responsible for representing the interests of everyone with a stake in the project and its resulting product.

Scrum
Not an acronym, but mechanisms in the game of rugby for getting an out-of-play ball back into play.

ScrumMaster
The person responsible for the Scrum process, its correct implementation, and the maximization of its benefits.

Sprint Backlog
A list of tasks that defines a Team's work for a Sprint. The list emerges during the Sprint. Each task identifies those responsible for doing the work and the estimated amount of work remaining on the task on any given day during the Sprint.

Sprint Backlog Task
One of the tasks that the Team or a Team member defines as required to turn committed Product Backlog items into system functionality.

Sprint Planning meeting
A one-day meeting time boxed to eight hours (for a four week Sprint) that initiates every Sprint. The meeting is divided into two four-hour segments, each also time boxed.. During the first four hours the Product Owner presents the highest priority Product Backlog to the team. The Team and Product Owner collaborate to help the Team determine how much Product Backlog it can turn into functionality during the upcoming Sprint. The Team commits to this at the end of the first four hours. During the second four hours of the meeting, the Team plans how it will meet this commitment by designing and then detailing its work as a plan in the Sprint Backlog.

Sprint Retrospective meeting
A time boxed three-hour meeting facilitated by the ScrumMaster at which the complete Team discusses the just-concluded Sprint and determines what could be changed that might make the next Sprint more enjoyable or productive.

Sprint Review meeting
A time-boxed four hour meeting at the end of every Sprint where the Team collaborates with the Product Owner and stakeholders on what just happened in the Sprint. This usually starts with a demonstration of completed Product Backlog items, a discussion of opportunities, constraints and findings, and a discussion of what might be the best things to do next (potentially resulting in Product Backlog changes). Only completed product functionality can be demonstrated.

Stakeholder
Someone with an interest in the outcome of a project, either because they have funded it, will use it, or will be affected by it.

Team
A cross-functional group of people that is responsible for managing themselves to develop an increment of product every Sprint.

Time box
A period of time that cannot be exceeded and within which an event or meeting occurs. For example, a Daily Scrum meeting is time boxed at fifteen minutes and terminates at the end of fifteen minutes, regardless. For meetings, it might last shorter. For Sprints, it lasts exactly that length.

Rolling out Agile at a large Enterprise

Gabrielle Benefield
Senior Director of Agile Development at Yahoo! Inc., 2007

Abstract

Yahoo! is a $50B company that has one of the largest Agile implementations in the world. The adoption of Scrum and Agile practices has been steadily growing over the past two years, and now encompasses more than 150 Yahoo! teams and more than 1500 people in the United States, Europe, and Asia-Pacific. The projects range from new product development such as Yahoo! Autos to heavy-duty infrastructure work on Yahoo! Mail which serves 250 million users each month around the globe.

Introduction

In the highly competitive Internet space, getting products to market quickly while being both flexible and adaptive to change is critical. Yahoo! needed a process that supported an Internet start-up culture within the structure of providing products and services to more than 500 million users worldwide.

Background

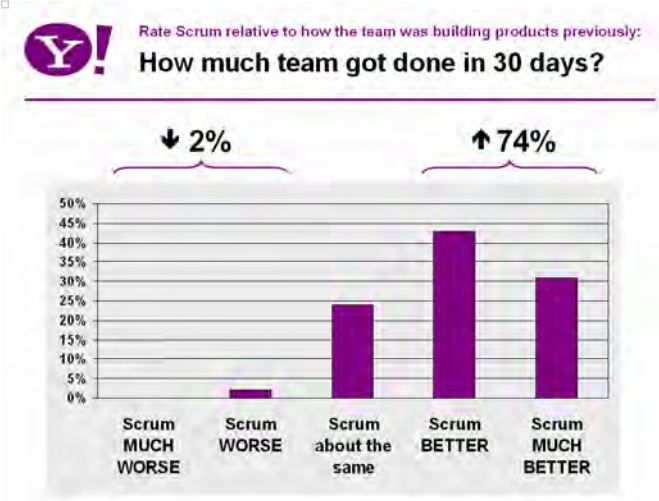
Yahoo! went from being a small start-up and grew to a large enterprise company quickly. The company still seems like a large start-up with the good and bad that comes with it. The things people liked about being a start-up was working closely with a small set of people, being able to quickly get products to market, the code base was relatively small and simple to work within and technical debt had not built up in it. The interdependencies between products are small and scaling for a small set of users is easy to deal with on the backend and in the application layer. Standardization in brand, user interface and tools is fairly straightforward. If you need something you usually know who to go to and how to find them to get things done. It is also very exciting as you ramp up quickly and the money starts rolling in. As a company grows it needs to deal with the complexity of many moving pieces, more people who you don't know, the logistics of seating, feeding and making thousands of employees happy. You are under intense public scrutiny, particularly if you are a Public company. Legal concerns heighten and if anything goes wrong the effects can be massive and broad reaching. No longer can you simply launch a product, multiple stakeholders need to be involved in the decision making, multiple properties may be affected if interconnected in the Yahoo! portal.

It is very hard to track down information as the size of the company grows. It appears to be a natural trend for start-ups that grow into large companies to hire in people with big company experience. Sometimes these people can add a lot of value, sometimes they can bring in overly bureaucratic processes that are at odds with the "just-get-things-done" start-up culture which drew in the employees in the first place. People who build systems from the ground up have a lot of passion and ownership and aren't always ready to share and the systems, platforms and tools put in place to service one set of needs, that of a small company no longer service hundreds of thousands, sometimes millions of users. In an attempt to make sense of scaling pain the knee jerk reaction is to often put in processes to manage and control software development. Often

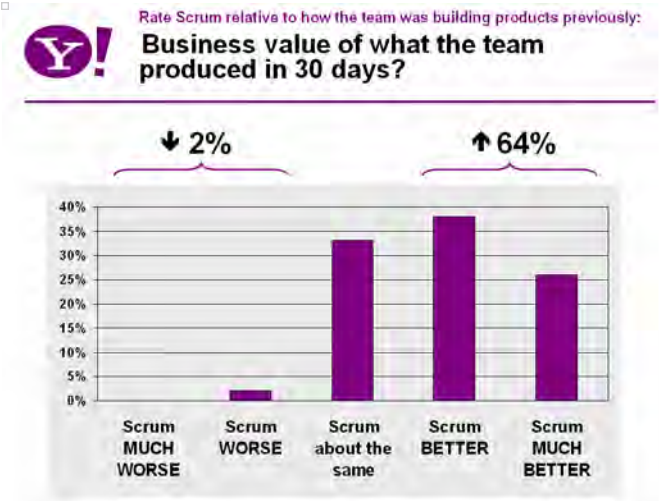
these look excellent on paper and in theory should work, unfortunately they can be at odds with the main ingredient in organizations, people.

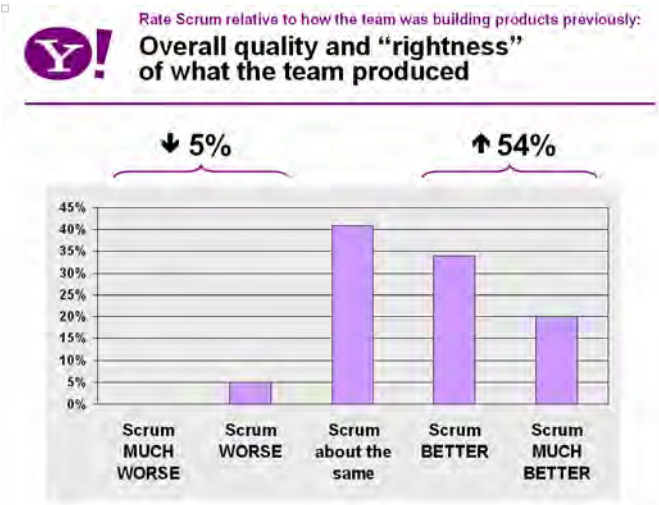
Yahoo! attempted to control the software development process and released a waterfall process called the "Product Development Process" in 2002. The process was rolled out globally and the use of it was mandatory. Unfortunately for the creators (or perhaps fortunately) a lot of teams simply ignored the process, or where they couldn't ignore it paid lip service and made it look like they adhered to the steps. The teams that did follow it found it was heavy, slowed them down and added little real value. Management felt like they were in control but the teams rebelled. There were some grass roots efforts in 2004 to try out some Agile practices such as Extreme Programming and Scrum. This was led by the team members or in one case by a smaller company (Stata Labs) that Yahoo! acquired. Tobias Mayer, an engineer on a team started a small grass roots movement to spread the word and found his way to the VP of Product Development at the time, Pete Deemer. Pete to his credit realized that the heavy weight process he had helped rollout was not succeeding as he had hoped and was curious about Agile. Tobias asked Ellen Salisbury, an engineering leader from Stata Labs to give an internal tech talk on her experiences. The talk was inspirational and piqued peoples interests. In a lucky confluence of events, Pete happened to contact Jeff Sutherland and Ken Schwaber when he was in the Bay Area (where Yahoo!'s main headquarters are located) on the same day as the executive team had an offsite dinner. Pete invited Jeff to be a guest speaker at the dinner to share his experiences with Scrum. The executive team was very inspired upon hearing his research that they decided to sponsor a pilot program on the spot. This led to the official rollout of the Scrum pilot program in February of 2005.

Pete and Tobias evangelized the benefits of Scrum amongst their contacts and managed to get four teams to volunteer to try Scrum for two months and participate in a survey to gather data about their experiences. The teams covered a broad set of products and services including the new Yahoo! Photos 3.0, a new backend for Yahoo! Mail, internal tools for managing small business sites and a media site re-design. A subset of team leads were sent to a Certified Scrum Master class with Ken Schwaber. The teams used a very standard out-of-the-box Scrum framework to address prioritization concerns, self-organization and teamwork, greater customer involvement and incremental product releases. At this stage little attention was put on technical practices as Scrum was seen as an easy first step to test the waters. At the end of their first month of using Scrum all the team members and their managers were invited to participate in an online survey to anonymously gather their feedback. The responders' received a custom printed Scrum t-shirt for participating, which also served a dual purpose to promote Scrum. The overall response rate was 71% (~85% for Scrum Pilot team members). The questions asked sought to track and collate information in the areas that Yahoo! wanted to improve upon which were mainly qualitative and focused on the human aspects of software development. The questions asked people to rate their experiences against their previous process.

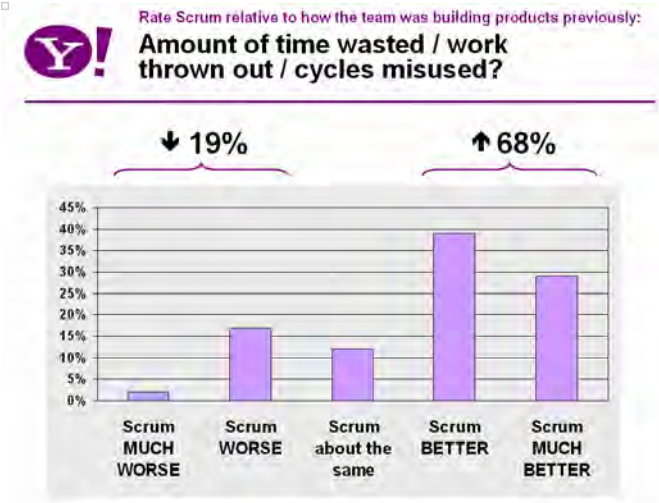

Rate Scrum relative to how the team was building products previously:
How much team got done in 30 days?
↓ 2%
↑ 74%
50%
45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
Scrum MUCH WORSE
Scrum WORSE
Scrum about the same
Scrum BETTER
Scrum MUCH BETTER

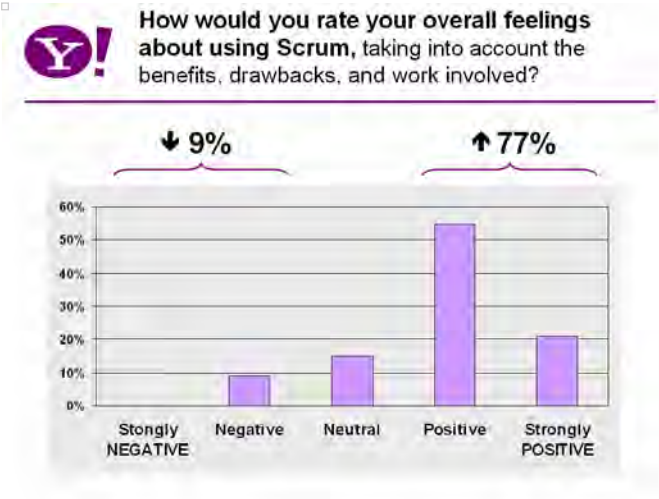

Rate Scrum relative to how the team was building products previously:
Clarity of goals / what the team was supposed to deliver?
↓ 6%
↑ 80%
45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
Scrum MUCH WORSE
Scrum WORSE
Scrum about the same
Scrum BETTER
Scrum MUCH BETTER

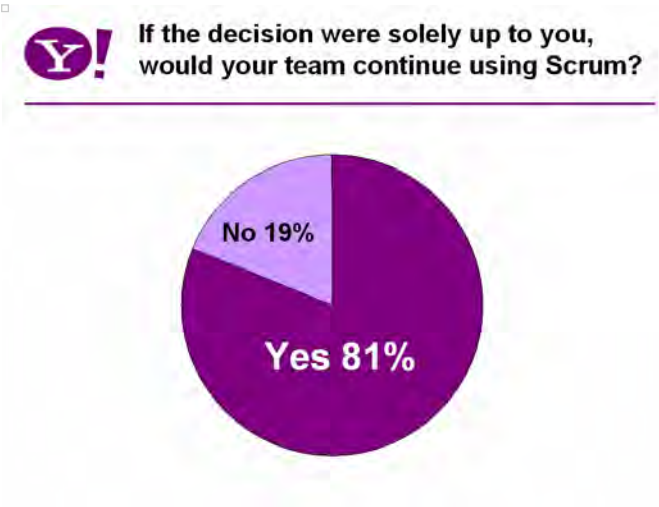

Rate Scrum relative to how the team was building products previously:
Business value of what the team produced in 30 days?
↓ 2%
↑ 64%
40%
35%
30%
25%
20%
15%
10%
5%
0%
Scrum MUCH WORSE
Scrum WORSE
Scrum about the same
Scrum BETTER
Scrum MUCH BETTER

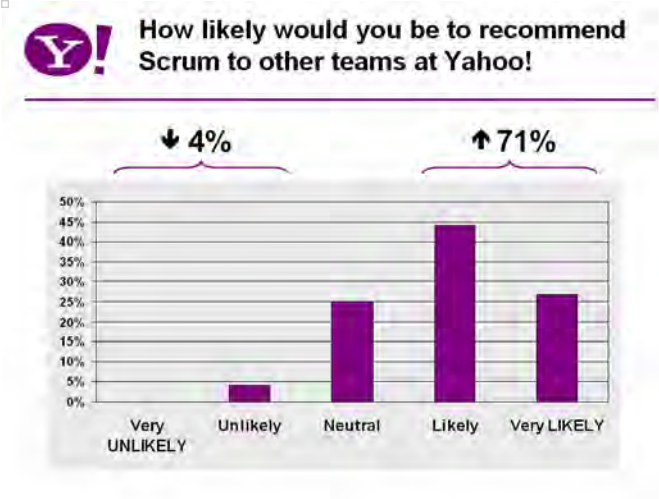

Rate Scrum relative to how the team was building products previously:
Overall quality and "rightness" of what the team produced
↓ 5%
↑ 54%
45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
Scrum MUCH WORSE
Scrum WORSE
Scrum about the same
Scrum BETTER
Scrum MUCH BETTER


Rate Scrum relative to how the team was building products previously:
Collaboration and cooperation within the team?
↓ 0%
↑ 89%
50%
45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
Scrum MUCH WORSE
Scrum WORSE
Scrum about the same
Scrum BETTER
Scrum MUCH BETTER


Rate Scrum relative to how the team was building products previously:
Amount of time wasted / work thrown out / cycles misused?
↓ 19%
↑ 68%
45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
Scrum MUCH WORSE
Scrum WORSE
Scrum about the same
Scrum BETTER
Scrum MUCH BETTER


How would you rate your overall feelings about using Scrum, taking into account the benefits, drawbacks, and work involved?
9%
77%
60%
50%
40%
30%
20%
10%
0%
Stongly NEGATIVE
Negative
Neutral
Positive
Strongly POSITIVE


If the decision were solely up to you, would your team continue using Scrum?
No 19%
Yes 81%


How likely would you be to recommend Scrum to other teams at Yahoo!
4%
71%
50%
45%
40%
35%
30%
25%
20%
15%
10%
5%
0%
Very UNLIKELY
Unlikely
Neutral
Likely
Very LIKELY

the feedback was positive; the teams liked the process and experience, and management saw positive results. Two years later we have over one hundred teams spread around the globe and we continue to grow rapidly.

The culture at Yahoo is very much like a large start-up. There is a constant stream of innovative ideas from every department, new product features are constantly released, and the company strives to be the first to market with new services, while meeting the needs of our users. The founders still work at the company, and have remained involved in day-to-day activities. They continue to instill in all employees a love for the culture and work. The company is committed to preserving the things that make Yahoo! great while putting in some process and practices to help teams deliver better products faster; this was the greatest challenge we faced when introducing Agile.

We started with Scrum, implementing small, cross-functional teams to address organizational issues, highlight business priorities and most importantly, create a collaborative environment. Next we added in Agile engineering practices and Lean fundamentals to deliver greater business value.

Kick-off

To kick off the program, we focused on building excitement internally, and motivating employees to get involved. We invited guest speakers like Ken Schwaber and Jeff Sutherland, the inventors of Scrum to address employees, and had a new employee give a talk on her experiences with Scrum at a previous company. We were also fortunate to have the VP of Product Development evangelize the benefits of Scrum to the top executives in the language that they could relate to.

Once we laid the ground work for the pilot program we experimented with an engagement model that allowed us to coach multiple teams efficiently. Where we had the bandwidth we would work closely with teams to get them up and running.

We found it worked well to start by spending some time preparing the Product Backlog with the Product Owner, then train the whole team together. A coach was assigned to lead the first Sprint planning meeting, stand-up, Sprint review and retrospective. For the second sprint we let the Scrum Master lead the team and we shadowed them. After that, depending on the team, we would do drop-ins and try to keep a good connection with the Product Owner and Scrum Master to help guide them through the early iterations. At this point, some teams were off and running, while others with difficult issues needed some additional coaching. We also found that teams unlearnt things over extended periods of time so we stayed in contact and continually re-engaged to lead some master retrospectives and give them objective advice and coaching.

We found an evolutionary approach to be far more successful than a revolutionary one for long term good results. At times we needed to push the envelope and take risks but trying to rip the band-aid off too quickly is dangerous and can lead to ultimate failure. For us, organizational change meant getting buy in at all levels, having people see results and be able to learn in a safe environment.

Management Support

Agile is all about experimentation and the ability to inspect and adapt as an empirical approach. At the end of the day, nothing really mattered but what employees actually experienced and all ideas were useless unless we executed on them and could prove that they worked. One thing that was key to the ongoing funding and success of the program was a quarterly survey. The survey was distributed to all the team members and managers, and we used the data to help us improve, worked with teams that most needed it, and distributed the data back to management (see examples of the questions and responses at the end of this report). Over time the survey became less useful as a way to compare Scrum with the old process as people only had Scrum experience at the company. We also found that after a couple of surveys people didn't want to do anymore, so long term data gathering is not effective using a survey alone.

Employee Support

We decided to keep the program voluntary and still do. It was agreed that for a process to be truly successful it needed to stand on its own merits. Although Agile was bought in from the top-down, the fact that the program was never mandated meant it had bottom-up support.
While we built relationships at all levels and marketed the successes to the management team, the real driving force was letting the word spread virally. The teams using Scrum spread the word about the process and people moving throughout the company seeded new teams. We leveraged the experiences of the people in the trenches to create a very effective promotion engine.

Feedback

We tried to keep a lot of transparency around the process and feedback we were receiving. We gained far more credibility by being open and letting people know that the process is not a silver bullet and acknowledging that change is hard. By being upfront with the challenges we were able to confront difficult issues and improve. We had panel discussions and "tech talks" from different Scrum teams to share their experiences, the challenges of transitioning to Agile and how they dealt with issues. A core tenet of Agile is around transparency and we felt this should also go for the methodology itself.

Top-down mandates that tried to enforce Scrum practices in a by-the-book fashion always backfired for us, as did teams that followed the practices so zealously they lost the forest for the trees. We did have teams that simply weren't ready or willing to use Agile and we had to respect that; we didn't want to force our coaches to be the process police and become part of the problem.

We didn't get too hung up on having the perfect tracking tools, training materials, coaching program, etc. in place. We made a lot of mistakes but we also improved quickly based on iterative feedback. Our philosophy dictated that it was better to make the flight than have our bags packed perfectly and still be waiting on the ground.

Roll-out

To kickoff an enterprise Agile rollout, we found it really helped to have people with real experience in the field. The foundation of the overall strategy was built on lessons learned and understanding how to deal with change. We built out a centralized team of coaches who were passionate and good at building relationships.

The team had a mixture of skills including Product management, QA, Design, Extreme Programming, Scrum and Lean. It was useful to have people with specializations in additional to generalized coaching so they could build bridges into different functional groups. Personality was also key. We needed people with passion and enthusiasm, as we were only as good as the relationships we could form. Having people who were overly zealous or abrasive would have quickly bought the program to a halt. One important aspect of hiring was to find people with strong skills in collaboration and building consensus.

Next we found that the best Agile champions were the people already in the teams, from all levels and disciplines. These people knew the context and the challenges of their particular situation and could adjust the process to meet their team's needs. Finding good people who really get it and training them up to help their own team is one of the differentiators, and is the only way to scale effectively in a large organization.

Challenges

Managers often feel left out when the team becomes more self-organized and don't know how to transition from the traditional command and control model to one of a strategic and supportive leader. They sometimes lash out or subvert the process out of fear. Where we came across people who are anti-Agile, we tried to get them to understand their changed role and to give them some responsibility. Training and coaching these people is worth the investment. We also did have to deal with the reality that not everyone is willing or can change and ultimately the new environment may no longer be a good fit for them anymore.
We wanted people to participate in the surveys but after a couple of rounds they were bored with having to fill out survey information all over again. So we offered free t-shirts. These weren't any old t-shirt, they were cool. We didn't even brand them as the sheer ambiguity was very appealing and people would be curious to know what the t-shirts meant and this would open some doors for us. We had people signing up to try Scrum just to get a t-shirt.

Another challenge we had and still have to some extent is to keep to the key tenets of Scrum while adapting to different contexts. Scrum provides an extremely flexible framework and how you apply it is an open-ended question.

We have a very strong and brilliant design group at Yahoo! Our products are heavily consumer focused so design is very important to us. The Designers initial reaction to Agile was similar to the way engineering architects react when faced with the idea that you don't design everything up front that you constantly re-factor and that requirements will change.

Working to understand the challenges and finding common ground helped improve the situation. There are some things that do fit with design thinking. Designers do want to adapt and work

incrementally. User stories that are focused on the customer are also warmly greeted. Lean thinking in keeping the features to a minimum and doing them well also strikes a positive note. We tried to be flexible, to listen to the design viewpoint and to help the whole team find a way to work together in a way that made sense to them. If the whole team was not able to find common ground that worked at the expense of key members the holistic team did not succeed.

We allowed the teams to find a way to work within the Agile framework that suited their context and needs, using Scrum as a flexible framework. We have teams that do overlap work within each iteration, where some design is done looking forward to the next iteration, some user testing of work completed in a previous iteration, and handoffs during the iteration occur. This may not look like pure Scrum but it works for teams developing consumer facing products and Ken Schwaber has always tried to get across the values rather than the rules of Scrum so we took this very much to heart. If we tried to enforce only working on tasks for the current iteration during the current iteration the designers would have mutinied and the team collaboration would have suffered. Again, Scrum is adaptive and if it works for people they keep doing it.

We have found though that the number one reason designers like Agile is the collaboration aspect. If the team spirit is strong and collaboration between team members is working they can overcome the logistical difficulties as they work together.

Training

It was and still is extremely challenging to get executives and senior managers to Scrum training due to their busy schedules, but it is also invaluable and worth the investment. One General Manager took the two day Certified Scrum Master class with his team and said it was a great experience. He got to hear the tough challenges and issues the team were facing while the team got a lot of insight into the business challenges and vision. This established a healthy base for ongoing two-way conversations. The manager bought a lot of credibility for investing time to sit and learn in the same room as everyone else.
Even though we had great internal coaches, we were very understaffed. We realized that having great people like Mike Cohn, Ken Schwaber, Jeff Sutherland, Mary and Tom Poppendieck was crucial to getting the program off the ground. It is worth spending the money on consultants if you lack internal expertise as they can ultimately save the company money if you apply them wisely.

Next time

The Agile Development team at Yahoo! approached the strategy with pragmatism and adaptability, and has experienced great success with the program. Nevertheless, there are always things you can change, including dedicating more resources and funding to the project, but until we could prove the process worked the business was not going to invest a lot. The whole process has been and still is all about learning and adapting as we go. The failures propelled us to new levels and it was important to allow teams to understand that failure is in itself an effective learning mechanism.

Fund the internal coaching team adequately
It would have been great to have the internal coaching team staffed adequately so we could get a good scaling strategy in place earlier. I was working by myself for a period of time, and the group only had two full time coaches consistently for the first year. We put new meaning to the term "lean". It took a long time to get more resources assigned to the central coaching team and this only came after financial analysis helped to prove each coach's value (*around 1.4 million dollars saved per year for each coach helping 10 teams be more productive*). This would have made a lot more teams a lot more successful and we could have scaled faster and better with additional resources.

Encourage deeper engagement from coaching staff
It would have been preferable to coach teams more intensely rather than being so broadly focused. Due to constrained resources and huge demand it was impossible to work as deeply with teams as we would have liked, and this showed very clearly in the survey results. *The teams with adequate coaching showed productivity increases of 2-3 times more than teams trying to work by themselves.*

There are teams that kicked off by attending a public class and had no follow-up coaching, due to either bandwidth restrictions or not realizing the value of it. We sometimes run into these teams or hear about them through the grapevine and find out they are not really doing Scrum at all but a hybrid that allows them to continue their dysfunctional practices while calling it Scrum. This is a major problem so now when teach classes we stress the importance of coaching.

It would have also been great to have solid engineering coaches available from day one to work with new teams, helping them set up build and test systems and introducing Agile engineering concepts. It is very challenging to deliver incremental products without good engineering discipline and this has definitely held back the productivity and quality of many of our teams.

Deeper management involvement
We could have pushed harder to get senior management to attend focused training. Having only a shallow understanding has lead to misunderstandings that could have been avoided if we had been able to do more of this. Again, in an ideal world with more resources we could have done more targeted training for management, product management, QA and design to better integrate them into the collective team spirit. We are currently working towards this.

Focus on the majority
It is a given that not everyone will be happy in every situation. We found in the process of introducing Scrum to Yahoo! that we did have people who were very negative towards our efforts. Fear, control and politics are constantly challenging and we had to simply realize we weren't always going to make friends. People will react to the changes and if they didn't, you would probably be telling them what they wanted to hear, not perhaps what they need to hear.

Conclusion

Although we have over 100 teams at Yahoo!, we still have a long way to go. Some days it feels like we are winning and Agile is spreading its love over the whole company, other days teams revert to bad practices and new blocks appear that feel impossible to break through. Some teams

are very Agile, others do mini-waterfalls and call it Agile. Change is difficult, and to change a company as large as Yahoo! sometimes feels like trying to steer the Titanic with a small paddle. We learned that patience is important, as is remembering that even the smallest of incremental improvements have a massive payoff when you do them at large scale.

Results from Scrum

The benefits of Scrum reported by teams come in various aspects of their experience. Once each quarter, we surveyed everyone at Yahoo! using Scrum (including Product Owners, Team Members, ScrumMasters, and the functional managers of those individuals) and ask them to compare Scrum to the approach they were using previously. Below are some results from our previous surveys:

• **Productivity:** 68% of respondents reported Scrum is better or much better (4 or 5 on a 5-point scale); 5% reported Scrum is worse or much worse (1 or 2 on a 5-point scale); 27% reported Scrum is about the same (3 on a 5-point scale).
• **Team Morale:** 52% of respondents reported Scrum is better or much better; 9% reported Scrum is worse or much worse; 39% reported Scrum is about the same.
• **Adaptability:** 63% of respondents reported Scrum is better or much better; 4% reported Scrum is worse or much worse; 33% reported Scrum is about the same.
• **Accountability:** 62% of respondents reported Scrum is better or much better; 6% reported Scrum is worse or much worse; 32% reported Scrum is about the same.
• **Collaboration and Cooperation:** 81% of respondents reported Scrum is better or much better; 1% reported Scrum is worse or much worse; 18% reported Scrum is about the same.
• **Team productivity increased an average a 37% increase, based on the estimates of the Product Owners.**
• **86% of team-members stated that they would continue using Scrum if the decision were solely up to them.**

Contact: Gabrielle Benefield (gbenefield@gmail.com)

Capturing Extreme Business Value: 1000% Annual Return on Investment in Scrum Trainers

Jeff Sutherland, Ph.D.
PatientKeeper, Inc., 2007

Abstract

In 2005, Jeff Sutherland worked together with Peter Deemer at Yahoo! to brief the Yahoo! senior management team on Scrum. After senior management made the decision to move forward with Scrum, a productivity analysis of rollout of Scrum at a previous large enterprise (IDX Systems, now GE Healthcare) was used to calculate an annual ROI of 1000% on a three year rollout of Scrum at Yahoo! After two years of deployment and Scrum rollout to over 100 teams the rate of return at Yahoo! for investment in each internal Scrum trainer was $1.4 based on training of 10 teams annually per trainer, or roughly 1000% return on investment . Teams coached by a Scrum trainer achieved 3-4 times the productivity gains of uncoached teams [26] .

Introduction

The internal rate of return on investment in Scrum training is quite high. Many companies have doubled the rate of software production on the average for all teams measured. Recently a CMMI Level 5 company cut the costs of software projects in half and reduced measured defects by 40% while still maintaining CMMI Level 5 compliance for all projects [19]. Even the best companies will radically improve performance by introducing Scrum and some will achieve far more than 1000% rate of return on investment in Scrum training.

This paper addresses the ROI on Scrum training for the average large company with thousands of employees and hundreds or thousands of developers. These companies have established heavyweight processes over many years that are bureaucratic and loaded with waste. While on the surface it would appear easy to provide substantial gains by eliminating the most obvious sources of inefficiency, introducing a radically new process company wide can be slow and painful. Scrum has a systematic continuous quality improvement process that identifies and prioritizes companywide impediments to progress.

IDX Systems (now GE Healthcare): Scaling Scrum for the First Time

During the summer of 1996, IDX Systems (now GE Healthcare) hired Jeff Sutherland as senior VP of engineering and product development. IDX had over 4,000 customers and was one of the largest US 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 organize the entire development group into an interlocking set of Scrums. While this was the first large development team to try this approach, the strategy has now been executed many times and documented by Ken Schwaber in "Scrum in the Enterprise" [27]. Every part of the organization was team based, including the management team, which included two vice presidents, a senior architect, and several directors. Front-line Scrums met daily. A Scrum of Scrums, which included the team leaders of each Scrum in a product line, met weekly, The management Scrum met monthly.

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

One of the most productive teams at IDX was the Web Framework team that built a web frontend infrastructure for all products. The infrastructure was designed to host all IDX applications, as well as seamlessly interoperate with end user or third party applications. The Web Framework was created by a distributed team with developers in Boston, Seattle, and Vermont who met by teleconference in a daily Scrum meeting. The geographic transparency of this model produced the same high performance as co-located teams and has become the signature of hyperproductive distributed/outsourced Scrums at Xebia in the Netherlands/India and Exigen Services in United States/Russia [28].

The quality of software of many of the hyperproductive Scrum teams can be extraordinarily high. The IDX Web Framework was first deployed in 1997 and in 2007 was selected as the core web technology for GE Healthcare systems. However, very few of the IDX software teams achieved the hyperproductive state. On the average, based on function point analysis by Capers Jones company, Software Productivity Research, IDX only achieved average productive gains of 240%, primarily due to loss of production because of large Scrum teams, up to 15 people in size. It is well understood today that these large teams cause significant loss in productivity and make it impossible to achieve linear scalability, one of the key features of well-executed Scrum implementations in the best Scrum companies.

Summary of Productivity Gains at IDX

The budget of the IDX development organization was almost $50M per year and this was sufficient size to a detailed analysis of baseline productivity before Scrum and productive gains generated by Scrum. An independent consulting firm, Software Productivity Research (SPR), was hired to do function point analysis of every IDX software product. Jeff Sutherland had worked with Capers Jones [29], the founder of SPR, during the original creation of Scrum and wanted to compare the productivity goals designed into the Scrum process with actual performance in the field. Some of the IDX products were sized at over 12000 function points,

the equivalent of over a million lines of Java or C# code and the smaller products were typically 5000-6000 function points. Applications were financial and clinical products for operating hospitals and independent physician groups at thousands of sites. Implementation platforms varied from Mumps to Cobol to Java and the latest Microsoft tools and languages available.

Function points were chosen as an industry standard measure that was independent of the software development language and environment. This was to ensure realistic comparison of productivity across technologies and development teams, as well as comparability with external industry data. External professional experts were hired to calculate function points in order to provide research quality data. Function points are not easy calculated by the average development team and not recommended as an operational strategy. Story points have emerged as industry best practice for measuring Agile development team velocity [30]. They are easily calculated and useful for release planning. However, they are not comparable across teams or across companies so were not suitable for research data on the first deployment of Scrum in a large enterprise.

At every release point for every product, the number of function points was recalculated to reflect the new features by Software Productivity Research consultants. The increase in function points was divided by person months for the fully burdened development teams including design, coding, testing, administrative, and management staff. Initial velocity of all development teams was industry average at 2-3 function points per staff month.

Some teams accelerated into a hyperproductive state using Scrum, achieving 5-10 times industry average performance. These teams were about 10% of the organization and achieved an average productivity increase of 666%. A small number of teams (less than 5%) experienced failures for either personnel or technology reasons. Occasionally, a team would not be able to work together effectively and was reorganized or disbanded, usually during the first Sprint. Less frequently, a high performance team had taken a calculated risk on new technologies (always approved by management) and technical failure of some Sprints was anticipated (even encouraged) in order to gain a technology lead in the market. There were no failures of large projects, only failure to deliver software in isolated Sprints or a short series of Sprints. In the case of team failures, the teams were always reformed. In the case of technology failures, efforts were redoubled in subsequent Sprints to overcome research and development challenges.

The remaining 85% of teams achieved an average velocity of 5-6 function points per staff month, averaging a 100% gain. The net productivity gain of all teams combined was 240%. This was viewed as a failure to achieve Toyota level performance. However, it was a good first start for enterprise wide deployment of Scrum.

It is important to note that these productivity gains were achieved at a sustainable pace with increased employee retention and enhanced ability to hire the best people in the software industry due to the high quality working environment provided by Scrum for developers. The hyperproductive teams were always the most spirited teams who loved their jobs and worked closely together like a professional sports team. Hyperproductivity is not achieved by working harder, but only by working better through intense communication, mutual support, and an "effortless" skill that makes hard things look easy. Think of Michael Jordan going up for a

basketball shot. The team has set him up and the shot often looks so smooth and easy it generates exhilaration in both the players and the spectators.

Chapter 2: The First Scrum

Scrum was derived from best practices in the Japanese auto and consumer products industry at Easel Corporation in 1993. The story of the first Scrum was published by the Cutter Agile Advisory Service outlining lessons learned on the first Scrum [21].

Senior management support is very helpful in implementing Scrum and at Easel, the CEO agreed to use Scrum for the first time on the most critical project in the company. Scrum was designed to take the toughest project and turn it into a success. It would never have begun without the support of the Easel CEO. How do you sell Scrum to management without any past experience or references to rely upon?

In 1993, the 1986 paper in the Harvard Business Review by Takeuchi and Nonaka [1] combined with the Coplien paper on the development of Quattro for Windows at Borland [2] triggered the first daily Scrum meetings and the monthly Sprint cycle. Japanese best practices in new product development at Honda and Fuji-Xerox reminded Takeuichi and Nonaka of the Scrum formation in Rugby. It would take more than these papers to convince a CEO under pressure to approve a new process he had never seen before. What were the key arguments?

Today, there are ROI analyses, experience reports, success stories, and lean manufacturing practices that help make a compelling case for selecting Scrum. Toyota has emerged as the leading Japanese example of lean product development. Many publications document Toyota's process which achieves 4 times the productivity and 12 times the quality of a typical U.S. competitor. This is what can be expected from a high quality Scrum implementation in software development. As a result OpenView Ventures Partners asked senior management teams in all of their portfolio companies to start learning about Scrum by reading the "Toyota Way" [13].

Agile Development: Lessons Learned from the First Scrum

Jeff Sutherland, Ph.D.
PatientKeeper, Inc., 2004

Introduction

Scrum for software development teams was developed at Easel Corporation in 1993, where we built the first object-oriented design and analysis (OOAD) tool that incorporated round-trip engineering. In a Smalltalk development environment, code was auto-generated from a graphic design tool and any changes to the code from the Smalltalk integrated development environment (IDE) were immediately reflected back into design.

Since the product was directed toward enterprise software development, we spent a lot of time analyzing best practices in software development methodologies.

Reviewing Software Development Processes

We realized we needed a development process that fit an enhanced version of rapid application development, where visualization of design could result immediately in working code. This led to an extensive review of both the literature and the real experience from leaders of hundreds of software development projects.

There were some key factors that influenced the introduction of Scrum at Easel Corporation. "Wicked Problems, Righteous Solutions" [31] reviewed the reasons why the waterfall approach to software development does not work.

- Requirements are not fully understood before the project begins,
- Users know what they want only after they see an initial version of the software,
- Requirements change often during the software construction process,
- And 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 and help resolve these challenges.

"All-at-Once" models assume that the creation of software is done by simultaneously working on requirements, analysis, design, coding, and testing, then delivering the entire system all at once. The simplest "All-at-Once" model is a single super-programmer creating and delivering an application from beginning to end. All aspects of the development process reside in one person's head. This is the fastest way to deliver a product that has good internal architectural consistency and is the "hacker" model of implementation. For example, in a project before the first Scrum, a single individual spent two years writing every line of code for the Matisse object database used

to drive $10B nuclear reprocessing plants worldwide. At less than 50,000 lines of code, the nuclear engineers said it was the fastest and most reliable database ever benchmarked for nuclear plants. Brooks has documented a variant of this approach called the Surgical Team, which IBM has shown to be their most productive software development process [32].

The Surgeon or super-programmer approach has the fatal flaw that there are at most one or two individuals even in a large company that can execute this model. For example, it took years for a leading team of developers to understand the conceptual elegance of the Matisse object server technology enough to maintain it. The single-programmer model does not scale well to large projects.

The next level of "All-at-Once" development is handcuffing two programmers together, as in pair programming in the eXtreme Programming paradigm [23]. Here, two developers working at the same terminal deliver a component of the system together. This has been shown to deliver better code (usability, maintainability, flexibility, extendibility) faster than two developers working individually [33]. The challenge is achieve a similar productivity effect with more than two people. What is the best way to work with multiple teams of people on large software projects?

Our scalable, team-based "All-at-Once" model was motivated by the Japanese approach to new product development. We were already using an iterative and incremental approach to building software [34]. It was implemented in slices in which 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 for setting up and managing a Scrum [1]. The idea of building a self-empowered team in which everyone had the global view of the product on a daily basis seemed like the right idea. The approach to managing the team, which had been so successful at Honda, Canon, and Fujitsu, also resonated with the systems thinking approach promoted by Professor Senge at MIT [15].

We were prodded into setting up the first Scrum meeting after reading Coplien's paper on Borland's development of Quattro Pro for Windows [2]. The Quattro team delivered one million lines of C++ code in 31 months with a 4-person staff that later grew to 8. This was about 1,000 lines of deliverable code per person per week, the most productive software 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.

Why the Easel CEO Supported the First Scrum

The primary driver for beginning the first Scrum was absolute commitment to a date, where failure would break the company. We had to guaranteed delivery of an innovative product to the market that would achieve rapid adoption.

Meeting with the CEO, I pointed out that he had been given plans for years that were supported by GANTT charts. He agreed no plan had ever delivered the required functionality on time.

Many delays had been extensive and hurt the company financially. Forecasted revenue on a major new product upgrade was millions of dollars a month so every month late cost the company millions in revenue. We could not afford late delivery again, as the company would operate at a loss for a quarter or more and damage to the stock price would be significant.

Further, I pointed out that in the past, he had no visibility on where the software was in the middle of the project. He had GANTT charts and reports that looked good on paper but never delivered the software on time. He had never seen a promised delivery date met and worse, he rarely discovered slippage until it was too late to reforecast company revenue.

I said to my CEO that if we adopt Scrum, we set the objectives at the beginning of a Sprint. It is the team's responsibility to figure out how to best meet those objectives. During the Sprint, no one can bother the team members. At the end of a Sprint, I added, we will have working code that can be demonstrated so he could see the progress being made. You can decide to ship anytime or do another Sprint to get more functionality. Visible working code will give you more confidence than extensive documentation with no operational system.

We committed to a fixed date six months out and planned for six monthly Sprints. The CEO agreed to proceed with the first software development Scrum. It took him about 60 seconds to decide. Little did he know how much of the future of global software development rested on that decision!

Scrum Basics

The first Scrum started with a half-day planning session that outlined the feature set we wanted to achieve in a six-month period, and then broke it into six pieces that were achievable in 30-day Sprints. This was the Product Backlog. For the first Sprint, the Product Backlog was transformed into development tasks that could be done in less than one day each, the first Sprint Backlog.

Short daily meetings were essential to drive the project with common mindshare. The three Scrum questions were used in the first Sprint. What did you do yesterday, what will you do today, and what is getting in your way? Daily meetings at Easel were disciplined in the way that we now understand as the Scrum pattern [25]. This radically altered the nature of the software development process. It allowed sharing of the state of software components so that development tasks, thought to take days, could often be accomplished in hours using someone else's code as a starting point.

One of the most interesting effect of Scrum on Easel's development environment was an observed "punctuated equilibrium" effect. This occurs in biological evolution when a species is stable for long periods of time and then undergoes a sudden jump in capability. During the long period of apparent stability, many internal changes in the organism are reconfigured that cannot be observed externally. When all pieces are in place to allow a significant jump in functionality, external change occurs suddenly. A fully integrated component design environment leads to unexpected, rapid evolution of a software system with emergent, adaptive properties resembling

the process of punctuated equilibrium observed in biological species. Sudden leaps in functionality resulted in earlier than expected delivery of software in the first Scrum [35].

This aspect of self-organization, the creators of Scrum now understand as a type of Set Based Engineering that is practiced at Toyota. Different developers were working on many components of the system and trying to evolve them as fast as possible. Decision on how to implement a task from a Sprint Backlog was delayed until the last possible moment. The most evolved component for the task was selected to absorb the new functionality.

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 at which it had to be slowed down by outnumbering developers with documentation and testing engineers. This *hyperproductive* state was seen in a many subsequent Scrums, although never as dramatic as the first one at Easel. It was a combination of (1) the skill of the team, (2) the flexibility of a Smalltalk development environment, and (3) the way we approached production prototypes that rapidly evolved into a deliverable product.

For example, a key to entering a hyperproductive state was not just the Scrum organizational pattern. We did constant component testing of topic areas, integration of packages, refactoring of selected parts of the system, and multiple builds per day. These activities have become key features of eXtreme Programming [36].

Adding the Set Based Engineering practice is now viewed as the “secret sauce” that turbocharged the process. So we are still learning lessons from the first Scrum. The magic happens when the Scrum process combines with good engineering practices and a sophisticated approach to product evolution. Set Based Engineering caused punctuated equilibrium and uncontrollably fast proliferation of functionality.

We held a demo every Friday during the first Scrum and brought development experts from other companies in to look at the product. As a result, our developers had to do demos to peers in other companies. This was one of the most powerful accelerators I have seen in software development. The outside experts would say, "That sucks, look at Borland's Product X to see how it should be done." Or "How could you possible have a dumb bug like that?"

The next week, everything would be fixed! The developers refused to be embarrassed again in front of their peers. The total transparency encouraged by Scrum was extended outside the company and MIT and Route 128 lead engineers self-organized to throw the Scrum into overdrive. This was very challenging to the Scrum team. Every week they felt they were not good enough and were depressed. I kept reminding them that to be world class, we had to repeatedly face defeat and triumph over it. We now understand from the President of Toyota that this repeated failure, along with inspecting and adapting, is a fundamental practice that allows persons and teams to move to a higher level of practice.

At the end of each month, the CEO got his demo. He could use the software himself and see it evolving. We then gave the software to the consulting group to use in prototyping consulting

projects. This gave us an incredible amount of feedback to incorporate into the Product Backlog, a list of features that are desirable to have in the software. At the beginning of each Sprint, the Product Backlog is reprioritized before transformation into development tasks. The Scrum adaptability to change allowed the CEO and the Product Owner to steer product development more effectively than other project management techniques.

Scrum Results

The CEO saw significant, step-by-step progress in each increment and agreed the software was ready to ship in the fifth increment. It had more functionality than expected in some areas and less in others. The sixth increment was primarily a packaging increment. We shipped on the day the product was scheduled to be shipped.

We gave a money-back guarantee that this new software would double developer productivity in the first month of use. It sold well until the Smalltalk market started to hit the wall in the mid-1990s, and became a model for Rational Rose development. The first ScrumMaster, John Scumniotales went on to lead the Rational Rose development team a few years later.

Everyone agreed that (1) Scrum could meet a deadline, (2) more functionality was achieved than expected, and (3) there would never be a return to a waterfall-type mentality because (1) waterfall could not predict, (2) it could not deliver on time, (3) it produced less functionality per developer unit of time, and (4) user satisfaction was terrible when the product was delivered, since waterfall approaches did not lend themselves to customer involvement or alteration of specifications required by rapidly changing market conditions.

Over the last decade, Scrum has emerged from humble beginnings to a movement involving tens of thousands of projects in hundreds of the leading software development companies worldwide. The process model used in the first Scrum 1993 is essentially the same as taught in Certified ScrumMaster courses in 2007.

The First Scrum: How Scrum provides energy, focus, clarity, and transparency to project teams developing complex systems

Interview with Dr. Jeff Sutherland, CTO of PatientKeeper, Inc.
EWork-Out Business Processes, May, 2006

"…The primary driver for beginning the first Scrum was absolute commitment to a date, where failure would break the company. The task: guaranteed delivery of an innovative product to the market that would achieve rapid adoption.

"In a meeting with the CEO, I noted that for years he had received project plans that were supported by Gantt charts. The CEO agreed that no plan had ever delivered the required functionality on time. Many delays had been extensive and hurt the company financially. "Forcasted revenue on a major new product upgrade was millions of dollars a month, so every month that a project was late cost the company millions in revenue. As the company would operated at a loss for a quarter or more and damage to the stock price would be significant, we could not afford to repeat this cycle.

"Further, I pointed out that the CEO had no view of the status of the software by the middle of the project. He had Gantt charts and reports that looked solid on paper but failed to deliver the software on time. He had never seen a promised delivery date met, and worse, he rarely discovered slippage until it was too late to reforecast company revenue.

"I told the CEO that in adopting Scrum, we set the objectives at the beginning of what Scrum refers to as a sprint. It is the teams responsibility to determine how to best meet those objectives. During the sprint, no one can bother team members with requests. At the end of a sprint, I added, working code that will be demonstrated, so you can see the progress made. You can decide to ship anytime or do another Sprint to get more functionality. Visible working code provides more confidence than extensive documentation with no operational system.

"In the case of this project, the date was six months out, and we established six sprints. The CEO agreed to proceed with the first software development Scrum.

"The first Scrum started with a half day planning session that outlined the feature set we wanted to achieve in a six month period. We then broke it into six pieces which were achievable in 30 day sprints. This was the product backlog. For the first sprint, the product backlog was transformed into development tasks that could be done in less than a day.

"Daily meetings allowed everyone on the project team to see the status of all aspects of the project in real time. This allowed the collective neural networks of the team's mind to fine-tune or redirect efforts on a daily basis to maximize throughput. The result was radical alteration of the software development process by allowing sharing of software resources. Development tasks thought to take days could often be accomplished in hours using someone else's code as a starting point.
…

“The meetings were kept short, typically under 30 minutes and discussion was restricted to the three SCRUM questions:

What did you do yesterday?
What will you do today?
What obstacles got in your way?
“By having every member of the team see every day what every other team member was doing, we could make progress by identifying work that could be improved by others. We received comments from one developer, for example, 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 at which it had to be slowed down by outnumbering developers with documentation and testing engineers. This *hyperproductive* state was seen in several subsequent Scrums, although never as dramatically as the first at Easel.
…
“At the end of each month, the CEO got a demo. He could use the software himself and see it work. We then gave the software to the consulting group to use in prototyping consulting projects. This provided an incredible amount of feedback to incorporate into the Scrum product backlog: a list of desirable features to include in the software. At the beginning of each sprint, product backlog is reprioritized before transformation into development tasks. The Scrum adaptability to change enabled the CEO to steer product development more effectively than other project management techniques.

“The CEO saw significant, step by step progress in each increment and he agreed that the product was ready to ship in the fifth increment. It had more functionality than expected in some areas and less in others. We shipped on the day it was scheduled to be shipped.
“Everyone agreed that first, Scrum could meet a deadline; second, more functionality was achieved than expected; and third, there would never be a return to [the old] mentality
“Over the past decade, Scrum has emerged from humble beginnings to a movement involving tens of thousands of projects in hundreds of the leading software development companies worldwide. Properly implemented, Scrum represents best business practice in some of the world's leading corporations.

“**It allows teams to operate close to the 'edge of chaos' to foster rapid system evolution, enforcing a simple set of rules for self-organization of software teams to produce systems with evolving architectures, aligning individual and organization objectives, creating a culture driven by performance, supporting shareholder value creation, achieving stable and consistent communication of performance at all levels, and enhancing individual development and quality of life.**”

Chapter 3: What is Scrum? The First Papers on the Scrum Development Process

Note: Best practices in Scrum have evolved over the years and may not reflected in this early paper. For example, at PatientKeeper, there is no closure phase to a project. All user acceptance testing, documentation and training is done as part of a Sprint and software goes into production at multiple large enterprises. This means the Sprint demo in advanced Scrum implementations is a live system with thousands of users at the end of every Sprint.

Many of the core principles taught in the Certified ScrumMaster training course are the ones created by the first Scrum team at Easel Corporation in 1993. Sprints, monthly iterations, daily meetings with three questions, impediments, and backlogs. The first Scrum also implemented all the eXtreme Programming engineering practices in some form two years before Kent Beck codified XP.

From 1995-2000, Jeff Sutherland chaired a series of workshops at the annual OOPSLA Conference on Business Object Design and Implementation [37]. Ken Schwaber agreed to write the first paper on Scrum for the OOPSLA'95 Workshop [38]. He observed the first Scrum in action and laid out the fundamental principles of operation. Over a decade later this paper is still one of the most widely read papers from OOPSLA Conferences. It continues to get more hits than any other paper at http://jeffsutherland.com/Scrum.

The concepts of complexity theory are introduced along with the important distinction between empirical and predictive processes. Business enterprises are complex adaptive systems and the software that runs them is rapidly becoming equally complex. Software systems evolve over time like biological systems. Concepts from artificial life [6] are informative as systems that are flexible evolve faster as flexibility increases up to the boundary of chaos. Using empirical process control to prevent chaotic behavior was shown to be the essence of Scrum in this paper.

Mike Beedle [25] led the Scrum process through elaboration as an organizational pattern supported by Ken Schwaber, Jeff Sutherland, and others (see Appendix I). Further information on the essence of Scrum can be found in *Agile Development with Scrum* by Ken Schwaber and Mike Beedle [17] with contributions from Jeff Sutherland that can be found in a paper later in this volume – Agile Can Scale: Inventing and Reinventing Scrum in Five Companies [39].

Over the last 13 years, Jeff Sutherland has used five companies as research laboratories for Scrum (Easel, VMARK, Individual, IDX, PatientKeeper) and Ken Schwaber has worked as a consultant in the last three of these companies where they have tested and refined Scrum together. Of continued interest is why a few Scrum teams enter the hyperproductive state like the first Scrum. This is clearly affected by the ***stages*** of maturity of Scrum implementations [40] along with the ***structure*** of deployment of Scrum teams [28].

Scrum Development Process

Ken Schwaber
Advanced Development Methods, Inc., 1995

Abstract

The stated, accepted philosophy for systems development is that the development process is a well understood approach that can be planned, estimated, and successfully completed. This has proven incorrect in practice. SCRUM assumes that the systems development process is an unpredictable, complicated process that can only be roughly described as an overall progression. SCRUM defines the systems development process as a loose set of activities that combines known, workable tools and techniques with the best that a development team can devise to build systems. Since these activities are loose, controls to manage the process and inherent risk are used. SCRUM is an enhancement of the commonly used iterative/incremental object-oriented development cycle.

KEY WORDS: Scrum SEI Capability-Maturity-Model Process Empirical

---

1. Introduction

In this paper we introduce a development process, Scrum, that treats major portions of systems development as a controlled black box. We relate this to complexity theory to show why this approach increases flexibility and ability to deal with complexity, and produces a system that is responsive to both initial and additionally occurring requirements.

Numerous approaches to improving the systems development process have been tried. Each has been touted as providing "significant productivity improvements." All have failed to produce dramatic improvements [41]. As Grady Booch noted, "We often call this condition the software crisis, but frankly, a malady that has carried on this long must be called normal."[42]

Concepts from industrial process control are applied to the field of systems development in this paper. Industrial process control defines processes as either "theoretical" (fully defined) or "empirical" (black box). When a black box process is treated as a fully defined process, unpredictable results occur [43]. A further treatment of this is provided in Appendix 1.

A significant number of systems development processes are not completely defined, but are treated as though they are. Unpredictability without control results. The Scrum approach treats these systems development processes as a controlled black box.
Variants of the Scrum approach for new product development with high performance small teams was first observed by Takeuchi and Nonaka [1][3] at Fuji-Xerox, Canon, Honda, NEC, Epson, Brother, 3M, Xerox, and Hewlett-Packard. A similar approach applied to software

development at Borland was observed by Coplien [2] to be the highest productivity C++ development project ever documented. More recently, a refined approach to the SCRUM process has been applied by Sutherland [44] to Smalltalk development and Schwaber [45] to Delphi development.

The Scrum approach is used at leading edge software companies with significant success. We believe Scrum may be appropriate for other software development organizations to realize the expected benefits from Object Oriented techniques and tools [46].

2. Overview

Our new approach to systems development is based on both defined and black box process management. We call the approach the Scrum methodology (see Takeuchi and Nonaka, 1986), after the Scrum in Rugby -- a tight formation of forwards who bind together in specific positions when a Scrumdown is called.

As will be discussed later, Scrum is an enhancement of the iterative and incremental approach to delivering object-oriented software initially documented by Pittman [47] and later expanded upon by Booch [48]. It may use the same roles for project staff as outlined by Graham [49], for example, but it organizes and manages the team process in a new way.

Scrum is a management, enhancement, and maintenance methodology for an existing system or production prototype. It assumes existing design and code which is virtually always the case in object-oriented development due to the presence of class libraries. Scrum will address totally new or re-engineered legacy systems development efforts at a later date.

Software product releases are planned based on the following variables:

- Customer requirements - how the current system needs enhancing.
- Time pressure - what time frame is required to gain a competitive advantage.
- Competition - what is the competition up to, and what is required to best them.
- Quality - what is the required quality, given the above variables.
- Vision - what changes are required at this stage to fulfill the system vision.
- Resource - what staff and funding are available.

These variables form the initial plan for a software enhancement project. However, these variables also change during the project. A successful development methodology must take these variables and their evolutionary nature into account.

3. Current Development Situation

Systems are developed in a highly complicated environment. The complexity is both within the development environment and the target environment. For example, when the air traffic control system development was initiated, three-tier client server systems and airline deregulation did not have to be considered. Yet, these environmental and technical changes occurred during the project and had to be taken into account within the system being built.

Environmental variables include:

- Availability of skilled professionals - the newer the technology, tools, methods, and domain, the smaller the pool of skilled professionals.

- Stability of implementation technology - the newer the technology, the lower the stability and the greater the need to balance the technology with other technologies and manual procedures.

- Stability and power of tools - the newer and more powerful the development tool, the smaller the pool of skilled professionals and the more unstable the tool functionality.

- Effectiveness of methods - what modeling, testing, version control, and design methods are going to be used, and how effective, efficient, and proven are they.

- Domain expertise - are skilled professionals available in the various domains, including business and technology.

- New features - what entirely new features are going to be added, and to what degree will these fit with current functionality.

- Methodology - does the overall approach to developing systems and using the selected methods promote flexibility, or is this a rigid, detailed approach that restricts flexibility.

- Competition - what will the competition do during the project? What new functionality will be announced or released.

- Time/Funding - how much time is available initially and as the project progresses? How much development funding is available.

- Other variables - any other factors that must be responded to during the project to ensure the success of the resulting, delivered system, such as reorganizations.

The overall complexity is a function of these variables:

complexity = f(development environment variables + target environment variables)

where these variables may and do change during the course of the project.

As the complexity of the project increases, the greater the need for controls, particularly the ongoing assessment and response to risk.

Attempts to model this development process have encountered the following problems:

- Many of the development processes are uncontrolled. The inputs and outputs are either unknown or loosely defined, the transformation process lacks necessary precision, and quality control is not defined. Testing processes are an example.

- An unknown number of development processes that bridge known but uncontrolled processes are unidentified. Detailed processes to ensure that a logical model contains adequate content to lead to a successful physical model are one such process.

- Environmental input (requirements) can only be taken into consideration at the beginning of the process. Complex change management procedures are required thereafter.

Attempts to impose a micro, or detailed, methodology model on the development process have not worked because the development process is still not completely defined. Acting as though the development process is defined and predictable, results in being unprepared for the unpredictable results.

Although the development process is incompletely defined and dynamic, numerous organizations have developed detailed development methodologies that include current development methods (structured, OO, etc.). The Waterfall methodology was one of the first such defined system development processes. A picture of the Waterfall methodology is shown in Figure 1.

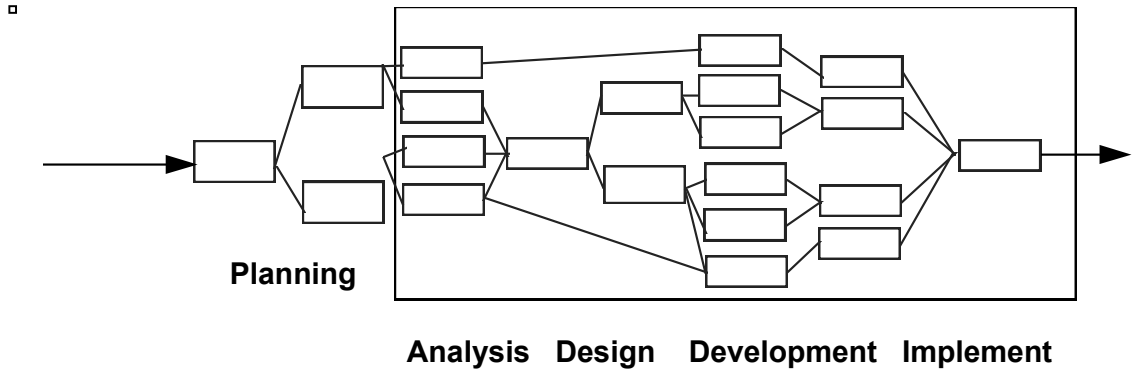


Figure 1 : Waterfall Methodology

Although the waterfall approach mandates the use of undefined processes, its linear nature has been its largest problem. The process does not define how to respond to unexpected output from any of the intermediate process.

Barry Boehm introduced a Spiral methodology to address this problem [50]. Each of the waterfall phases is ended with a risk assessment and prototyping activity. The Spiral methodology is shown in Figure 2.

The Spiral methodology "peels the onion," progressing through "layers" of the development process. A prototype lets users determine if the project is on track, should be sent back to prior phases, or should be ended. However, the phases and phase processes are still linear.

Requirements work is still performed in the requirements phase, design work in the design phase, and so forth, with each of the phases consisting of linear, explicitly defined processes.

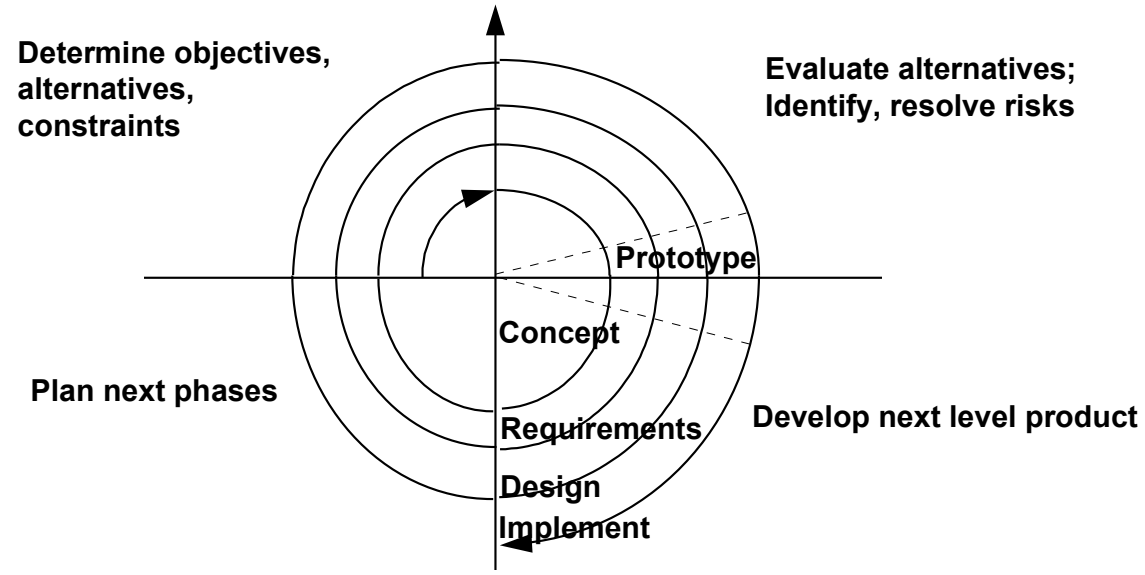


Figure 2 : Spiral Methodology

The Iterative methodology improves on the Spiral methodology. Each iteration consists of all of the standard Waterfall phases, but each iteration only addresses one set of parsed functionality. The overall project deliverable has been partitioned into prioritized subsystems, each with clean interfaces. Using this approach, one can test the feasibility of a subsystem and technology in the initial iterations. Further iterations can add resources to the project while ramping up the speed of delivery. This approach improves cost control, ensures delivery of systems (albeit subsystems), and improves overall flexibility. However, the Iterative approach still expects that the underlying development processes are defined and linear. See Figure 3.

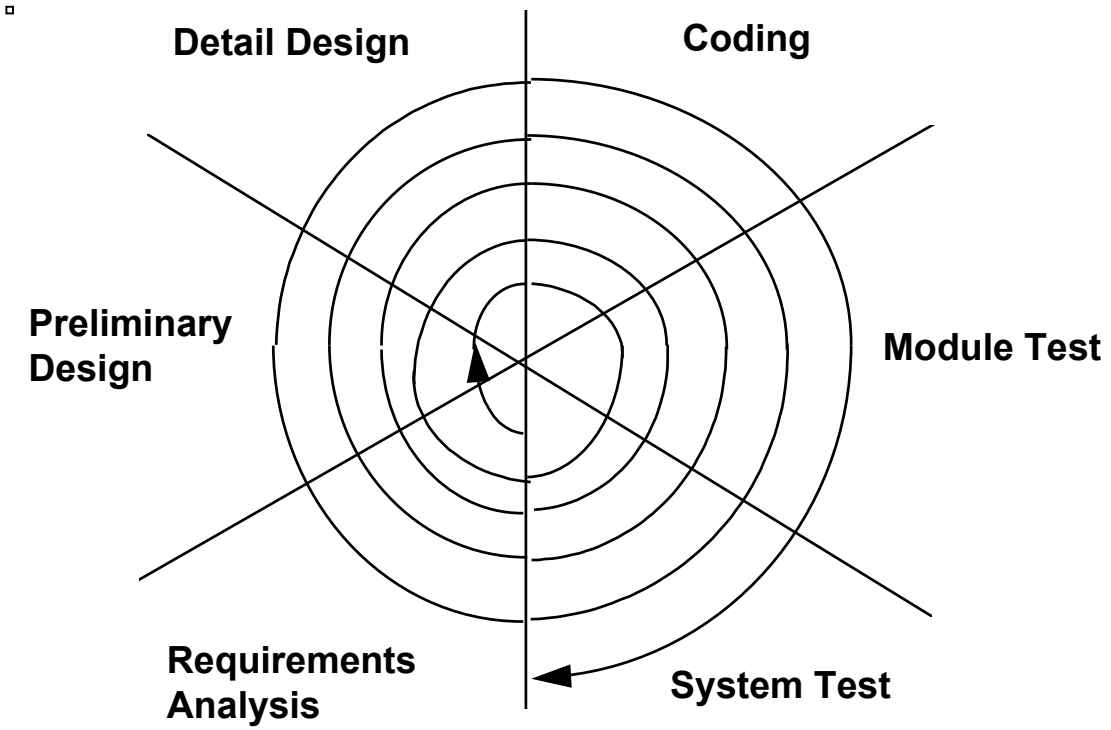


Figure 3 : Iterative Methodology

Given the complex environment and the increased reliance on new "state-of-the-art" systems, the risk endured by system development projects has increased and the search for mechanisms to handle this risk has intensified.

One can argue that current methodologies are better than nothing. Each improves on the other. The Spiral and Iterative approaches implant formal risk control mechanisms for dealing with unpredictable results. A framework for development is provided.

However, each rests on the fallacy that the development processes are defined, predictable processes. But unpredictable results occur throughout the projects. The rigor implied in the development processes stifles the flexibility needed to cope with the unpredictable results and respond to a complex environment.

Despite their widespread presence in the development community, our experience in the industry shows that people do not use the methodologies except as a macro process map, or for their detailed method descriptions.

The following graph demonstrates the current development environment, using any of the Waterfall, Spiral or Iterative processes. As the complexity of the variables increase even to a moderate level, the probability of a “successful” project quickly diminishes (a successful project is defined as a system that is useful when delivered). See Figure 4.

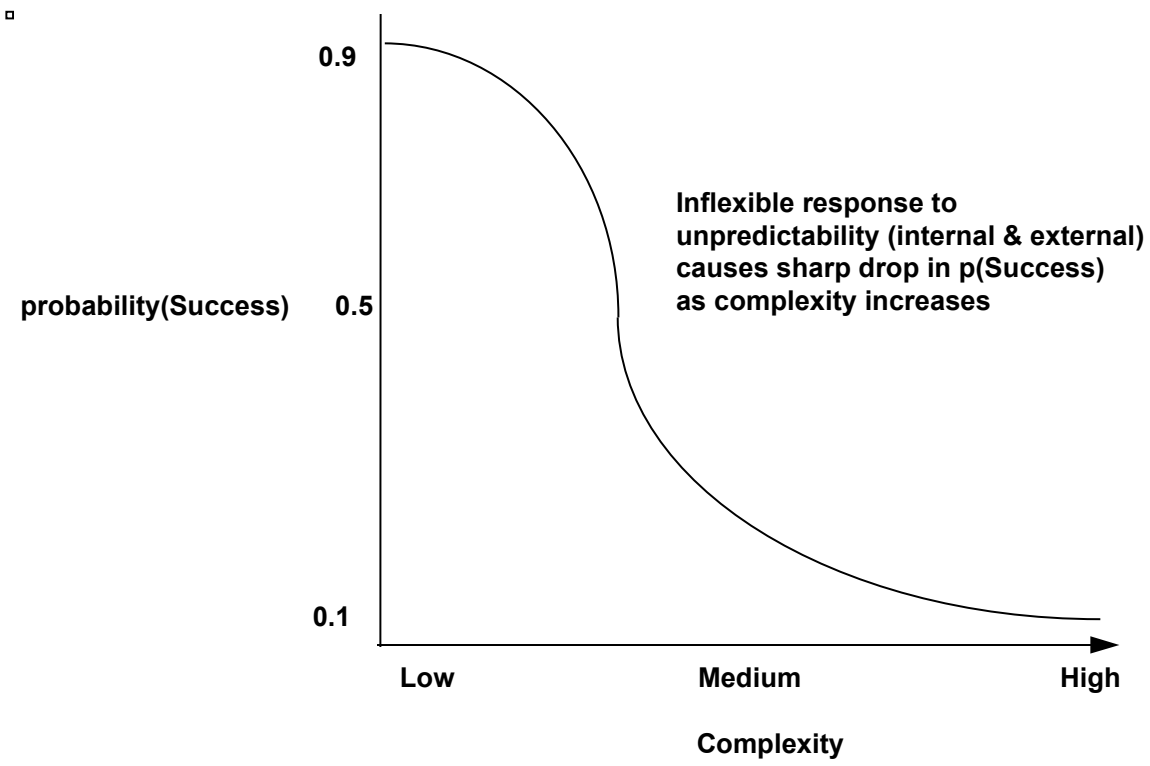


Figure 4: Defined Process Risk/Complexity Graph

4. Scrum Methodology

The system development process is complicated and complex. Therefore maximum flexibility and appropriate control is required. Evolution favors those that operate with maximum exposure to environmental change and have maximized flexibility. Evolution deselects those who have insulated themselves from environmental change and have minimized chaos and complexity in their environment.

An approach is needed that enables development teams to operate adaptively within a complex environment using imprecise processes. Complex system development occurs under rapidly changing circumstances. Producing orderly systems under chaotic circumstances requires maximum flexibility. The closer the development team operates to the edge of chaos, while still maintaining order, the more competitive and useful the resulting system will be. Langton has modeled this effect in computer simulations [6] and his work has provided this as a fundamental theorem in complexity theory.

Methodology may well be the most important factor in determining the probability of success. Methodologies that encourage and support flexibility have a high degree of tolerance for changes in other variables. With these methodologies, the development process is regarded as unpredictable at the onset, and control mechanisms are put in place to manage the unpredictability.

If we graph the relationship between environmental complexity and probability of success with a flexible methodology that incorporates controls and risk management, the tolerance for change is more durable. See Figure 5.

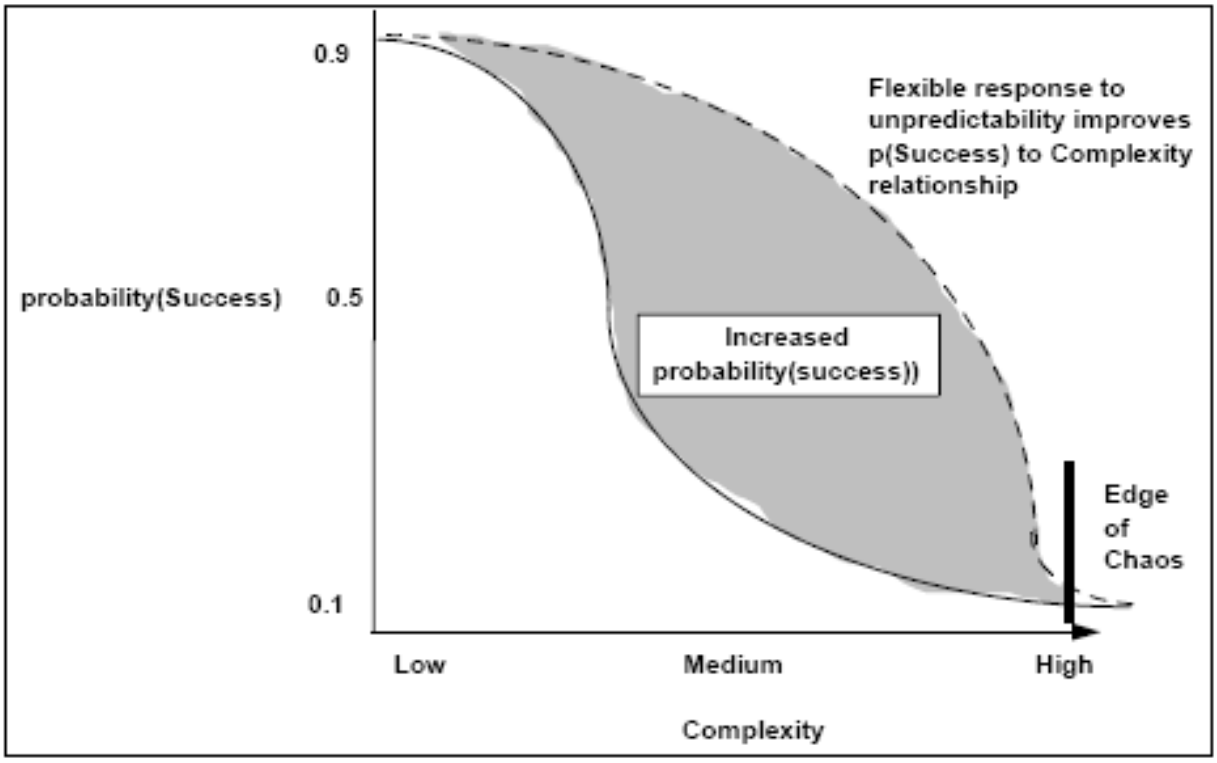


Figure 5: Risk/Complexity Comparison Graph

Figures 4 and 5 reflect software development experiences at ADM, Easel, VMARK, Borland and virtually every other developer of "packaged" software. These organizations have embraced risk and environmental complexity during development projects. Increased product impact, successful projects, and productivity gains were experienced. The best possible software is built.

Waterfall and Spiral methodologies set the context and deliverable definition at the start of a project. Scrum and Iterative methodologies initially plan the context and broad deliverable definition, and then evolve the deliverable during the project based on the environment. Scrum acknowledges that the underlying development processes are incompletely defined and uses control mechanisms to improve flexibility.

The primary difference between the defined (waterfall, spiral and iterative) and empirical (Scrum) approach is that the Scrum approach assumes that the analysis, design, and development processes in the Sprint phase are unpredictable. A control mechanism is used to manage the unpredictability and control the risk. Flexibility, responsiveness, and reliability are the results. See Figure 6.

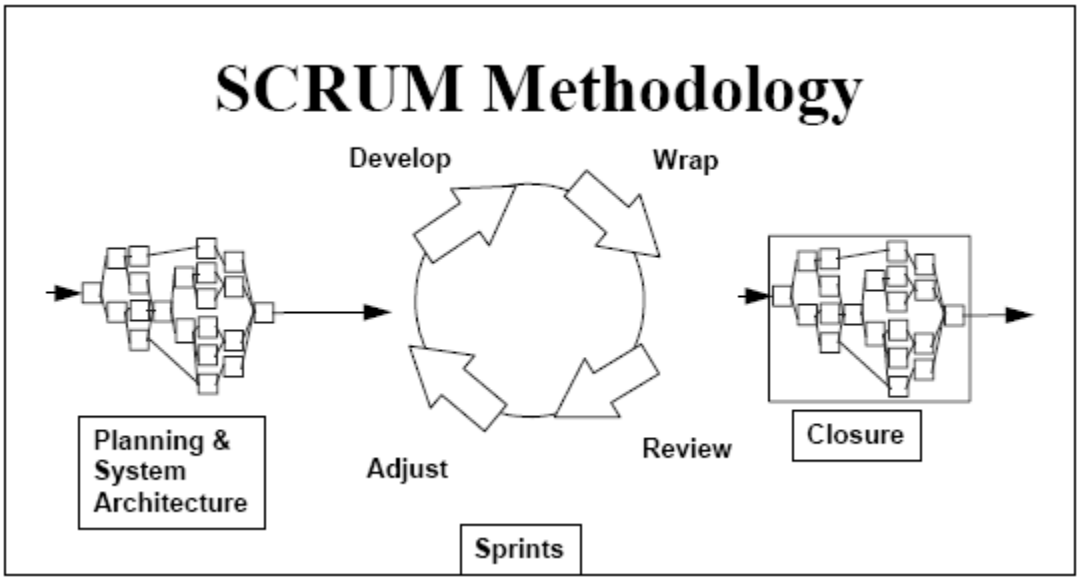


Figure 6 : Scrum Methodology

Characteristics of Scrum methodology are :

- The first and last phases (Planning and Closure) consist of defined processes, where all processes, inputs and outputs are well defined. The knowledge of how to do these processes is explicit. The flow is linear, with some iterations in the planning phase.

- The Sprint phase is an empirical process. Many of the processes in the sprint phase are unidentified or uncontrolled. It is treated as a black box that requires external controls. Accordingly, controls, including risk management, are put on each iteration of the Sprint phase to avoid chaos while maximizing flexibility.

- Sprints are nonlinear and flexible. Where available, explicit process knowledge is used; otherwise tacit knowledge and trial and error is used to build process knowledge. Sprints are used to evolve the final product.

- The project is open to the environment until the Closure phase. The deliverable can be changed at any time during the Planning and Sprint phases of the project. The project remains open to environmental complexity, including competitive, time, quality, and financial pressures, throughout these phases.

- The deliverable is determined during the project based on the environment.

The table in Figure 7 compares the primary Scrum characteristics to those of other methodologies.

	Waterfall	Spiral	Iterative	Scrum
Defined processes	Required	Required	Required	Planning & Closure only
Final product	Determined during planning	Determined during planning	Set during project	Set during project
Project cost	Determined during planning	Partially variable	Set during project	Set during project
Completion date	Determined during planning	Partially variable	Set during project	Set during project
Responsiveness to environment	Planning only	Planning primarily	At end of each iteration	**Throughout**
Team flexibility, creativity	Limited - cookbook approach	Limited - cookbook approach	Limited - cookbook approach	**Unlimited during iterations**
Knowledge transfer	Training prior to project	Training prior to project	Training prior to project	**Teamwork during project**
Probability of success	Low	Medium low	Medium	**High**

Figure 7 : Methodology Comparison

4.1 Scrum Phases

Scrum has the following groups of phases:

4.1.1. Pregame

- Planning : Definition of a new release based on currently known backlog, along with an estimate of its schedule and cost. If a new system is being developed, this phase consists of both conceptualization and analysis. If an existing system is being enhanced, this phase consists of limited analysis.
- Architecture : Design how the backlog items will be implemented. This phase includes system architecture modification and high level design.

4.1.2. Game

- Development Sprints : Development of new release functionality, with constant respect to the variables of time, requirements, quality, cost, and competition. Interaction with these variables defines the end of this phase. There are multiple, iterative development sprints, or cycles, that are used to evolve the system.

4.1.3. Postgame

- Closure : Preparation for release, including final documentation, pre-release staged testing, and release.

Figure 8: Scrum Methodology

4.2 Phase Steps

Each of the phases has the following steps:

4.2.1. Planning

- Development of a comprehensive backlog list.
- Definition of the delivery date and functionality of one or more releases.
- Selection of the release most appropriate for immediate development.
- Mapping of product packets (objects) for backlog items in the selected release.
- Definition of project team(s) for the building of the new release.
- Assessment of risk and appropriate risk controls.
- Review and possible adjustment of backlog items and packets.
- Validation or reselection of development tools and infrastructure.
- Estimation of release cost, including development, collateral material, marketing, training, and rollout.
- Verification of management approval and funding.

4.2.2. Architecture/High Level Design

- Review assigned backlog items.
- Identify changes necessary to implement backlog items.
- Perform domain analysis to the extent required to build, enhance, or update the domain models to reflect the new system context and requirements.
- Refine the system architecture to support the new context and requirements.
- Identify any problems or issues in developing or implementing the changes
- Design review meeting, each team presenting approach and changes to implement each backlog item. Reassign changes as required.

4.2.3. Development (Sprint)

The Development phase is an iterative cycle of development work. The management determines that time, competition, quality, or functionality are met, iterations are completed and the closure phase occurs. This approach is also known as Concurrent Engineering. Development consists of the following macro processes:

- Meeting with teams to review release plans.
- Distribution, review and adjustment of the standards with which the product will conform.
- Iterative Sprints, until the product is deemed ready for distribution.

A Sprint is a set of development activities conducted over a pre-defined period, usually one to four weeks. The interval is based on product complexity, risk assessment, and degree of oversight desired. Sprint speed and intensity are driven by the selected duration of the Sprint. Risk is assessed continuously and adequate risk controls and responses are put in place. Each Sprint consists of one or more teams performing the following:

- Develop: Defining changes needed for the implementation of backlog requirements into packets, opening the packets, performing domain analysis, designing, developing, implementing, testing, and documenting the changes. Development consists of the micro process of discovery, invention, and implementation.
- Wrap: Closing the packets, creating a executable version of changes and how they implement backlog requirements.
- Review: All teams meeting to present work and review progress, raising and resolving issues and problems, adding new backlog items. Risk is reviewed and appropriate responses defined.
- Adjust: Consolidating the information gathered from the review meeting into affected packets, including different look and feel and new properties.

Each Sprint is followed by a review, whose characteristics are :

- The whole team and product management are present and participate.
- The review can include customers, sales, marketing and others.
- Review covers functional, executable systems that encompass the objects assigned to that team and include the changes made to implement the backlog items.
- The way backlog items are implemented by changes may be changed based on the review.
- New backlog items may be introduced and assigned to teams as part of the review, changing the content and direction of deliverables.
- The time of the next review is determined based on progress and complexity. The Sprints usually have a duration of 1 to 4 weeks.

4.2.4. Closure

When the management team feels that the variables of time, competition, requirements, cost, and quality concur for a new release to occur, they declare the release "closed" and enter this phase. The closure phase prepares the developed product for general release. Integration, system test, user documentation, training material preparation, and marketing material preparation are among closure tasks.

4.3. Scrum Controls

Operating at the edge of chaos (unpredictability and complexity) requires management controls to avoid falling into chaos. The Scrum methodology embodies these general, loose controls, using OO techniques for the actual construction of deliverables.

Risk is the primary control. Risk assessment leads to changes in other controls and responses by the team.

Controls in the Scrum methodology are :

- Backlog: Product functionality requirements that are not adequately addressed by the current product release. Bugs, defects, customer requested enhancements, competitive product functionality, competitive edge functionality, and technology upgrades are backlog items.

- Release/Enhancement: backlog items that at a point in time represent a viable release based on the variables of requirements, time, quality, and competition.

- Packets: Product components or objects that must be changed to implement a backlog item into a new release.

- Changes: Changes that must occur to a packet to implement a backlog item.

- Problems: Technical problems that occur and must be solved to implement a change.

- Risks: risks that affect the success of the project are continuously assessed and responses planned. Other controls are affected as a result of risk assessment.

- Solutions: solutions to the problems and risks, often resulting in changes.

- Issues: Overall project and project issues that are not defined in terms of packets, changes and problems.

These controls are used in the various phases of Scrum. Management uses these controls to manage backlog. Teams use these controls to manage changes, problems. Both management and teams jointly manage issues, risks, and solutions. These controls are reviewed, modified, and reconciled at every Sprint review meeting.

4.4 Scrum Deliverables

The delivered product is flexible. Its content is determined by environment variables, including time, competition, cost, or functionality. The deliverable determinants are market intelligence, customer contact, and the skill of developers. Frequent adjustments to deliverable content occur during the project in response to environment. The deliverable can be determined anytime during the project.

4.5 Scrum Project Team

The team that works on the new release includes full time developers and external parties who will be affected by the new release, such as marketing, sales, and customers. In traditional

release processes, these latter groups are kept away from development teams for fear of over-complicating the process and providing "unnecessary" interference. The Scrum approach, however, welcomes and facilitates their controlled involvement at set intervals, as this increases the probability that release content and timing will be appropriate, useful, and marketable.

The following teams are formed for each new release:

Management: Led by the Product Manager, it defines initial content and timing of the release, then manages their evolution as the project progresses and variables change. Management deals with backlog, risk, and release content.

Development teams: Development teams are small, with each containing developers, documenters and quality control staff. One or more teams of between three and six people each are used. Each is assigned a set of packets (or objects), including all backlog items related to each packet. The team defines changes required to implement the backlog item in the packets, and manages all problems regarding the changes. Teams can be either functionally derived (assigned those packets that address specific sets of product functionality) or system derived (assigned unique layers of the system). The members of each team are selected based on their knowledge and expertise regarding sets of packets, or domain expertise.

4.6 Scrum Characteristics

The Scrum methodology is a metaphor for the game of Rugby. Rugby evolved from English football (soccer) under the intense pressure of the game:

> *Rugby student William Webb Ellis, 17, inaugurates a new game whose rules will be codified in 1839. Playing soccer for the 256-year-old college in East Warwickshire, Ellis sees that the clock is running out with his team behind so he scoops up the ball and runs with it in defiance of the rules.*
> The People's Chronology, Henry Holt and Company, Inc. Copyright © 1992.

Scrum projects have the following characteristics:

- Flexible deliverable - the content of the deliverable is dictated by the environment.
- Flexible schedule - the deliverable may be required sooner or later than initially planned.
- Small teams - each team has no more than 6 members. There may be multiple teams within a project.
- Frequent reviews - team progress is reviewed as frequently as environmental complexity and risk dictates (usually 1 to 4 week cycles). A functional executable must be prepared by each team for each review.
- Collaboration - intra and inter-collaboration is expected during the project.

- Object Oriented - each team will address a set of related objects, with clear interfaces and behavior.

The Scrum methodology shares many characteristics with the sport of Rugby:

- The context is set by playing field (environment) and Rugby rules (controls).
- The primary cycle is moving the ball forward.
- Rugby evolved from breaking soccer rules - adapting to the environment.

The game does not end until environment dictates (business need, competition, functionality, timetable).

5. Advantages of the Scrum Methodology

Traditional development methodologies are designed only to respond to the unpredictability of the external and development environments at the start of an enhancement cycle. Such newer approaches as the Boehm spiral methodology and its variants are still limited in their ability to respond to changing requirements once the project has started.

The Scrum methodology, on the other hand, is designed to be quite flexible throughout. It provides control mechanisms for planning a product release and then managing variables as the project progresses. This enables organizations to change the project and deliverables at any point in time, delivering the most appropriate release.

The Scrum methodology frees developers to devise the most ingenious solutions throughout the project, as learning occurs and the environment changes.

Small, collaborative teams of developers are able to share tacit knowledge about development processes. An excellent training environment for all parties is provided.

Object Oriented technology provides the basis for the Scrum methodology. Objects, or product features, offer a discrete and manageable environment. Procedural code, with its many and intertwined interfaces, is inappropriate for the Scrum methodology. Scrum may be selectively applied to procedural systems with clean interfaces and strong data orientation.

6. Scrum Project Estimating

Scrum projects can be estimated using standard function point estimating. However, it is advisable to estimate productivity at approximately twice the current metric. The estimate is only for starting purposes, however, since the overall timetable and cost are determined dynamically in response to the environmental factors.

Our observations have led us to conclude that Scrum projects have both velocity and acceleration. In terms of functions delivered, or backlog items completed:

- initial velocity and acceleration are low as infrastructure is built/modified
- as base functionality is put into objects, acceleration increases
- acceleration decreases and velocity remains sustainably high

Further development in metrics for empirical processes is required.

7. System Development Methodologies : Defined or Empirical

System development is the act of creating a logical construct that is implemented as logic and data on computers. The logical construct consists of inputs, processes, and outputs, both macro (whole construct) and micro (intermediate steps within whole construct). The whole is known as an implemented system.

Many artifacts are created while building the system. Artifacts may be used to guide thinking, check completeness, and create an audit trail. The artifacts consist of documents, models, programs, test cases, and other deliverables created prior to creating the implemented system. When available, a *metamodel* defines the semantic content of model artifacts. *Notation* describes the graphing and documentation conventions that are used to build the models.

The approach used to develop a system is known as a *method*. A *method* describes the activities involved in defining, building, and implementing a system; a *method* is a framework. Since a *method* is a logical process for constructing systems (process), it is known as a *metaprocess* (a process for modeling processes).

A *method* has micro and macro components. The macro components define the overall flow and time-sequenced framework for performing work. The micro components include *general design rules*, *patterns* and rules of *thumb*. *General design rules* state properties to achieve or to avoid in the design or general approaches to take while building a system. *Patterns* are solutions that can be applied to a type of development activity; they are solutions waiting for problems that occur during an activity in a method. **Rules of thumb** consist of a general body of hints and tips.

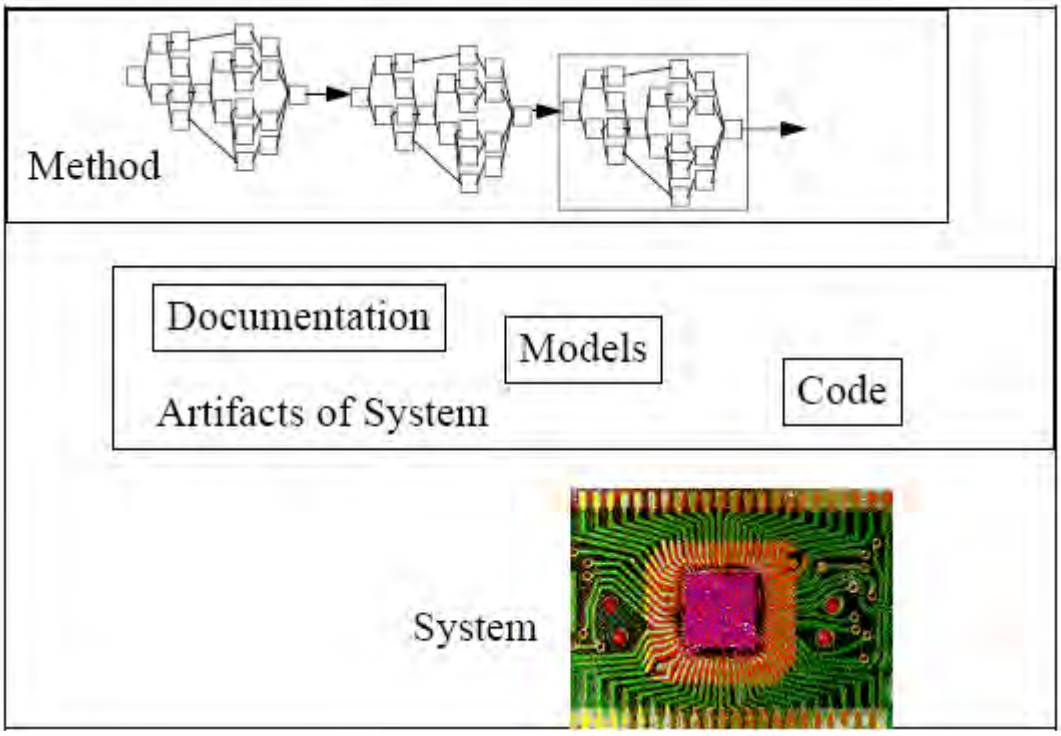


Figure 9: Relationship between the Method, the Artifacts, and the System

Applying concepts from industrial process control to the field of systems development, methods can be categorized as either "theoretical" (fully defined) or "empirical" (black box).

Correctly categorizing systems development methods is critical. The appropriate structure of a *method* for building a particular type of system depends on whether the *method* is theoretical or empirical.

Models of *theoretical processes* are derived from *first principles*, using material and energy balances and fundamental laws to determine the model. For a systems development *method* to be categorized as theoretical, it must conform to this definition.

Models of *empirical processes* are derived categorizing observed inputs and outputs, and defining controls that cause them to occur within prescribed bounds. Empirical process modeling involves constructing a process model strictly from experimentally obtained input/output data, with no recourse to any laws concerning the fundamental nature and properties of the system. No *a priori* knowledge about the process is necessary (although it can be helpful); a system is treated like a black box.

Primary characteristics of both theoretical and empirical modeling are detailed in Figure 10.

Theoretical Modeling	**Empirical Modeling**
1. Usually involves fewer measurements; requires experimentation only for the	Requires extensive measurements, because it relies entirely on experimentation for the

estimation of unknown model parameters.	model development.
2. Provides information about the internal state of the process.	Provides information only about that portion of the process which can be influenced by control action.
3. Promotes fundamental understanding of the internal workings of the process.	Treats the process like a "black box."
4. Requires fairly accurate and complete process knowledge.	Requires no such detailed knowledge; only that output data be obtainable in response to input changes.
5. Not particularly useful for poorly understood and/or complex processes.	Quite often proves to be the only alternative for modeling the behavior of poorly understood and/or complex processes.
6. Naturally produces both linear and nonlinear process models.	Requires special methods to produce nonlinear models.

Figure 10: Theoretical vs. empirical modeling

Upon inspection, we assert that the systems development process is empirical:

- Applicable first principles are not present
- The process is only beginning to be understood
- The process is complex
- The process is changing

Most methodologists agree with this assertion; "...you can't expect a *method* to tell you everything to do. Writing software is a creative process, like painting or writing or architecture... ... (*a method*) supplies a framework that tells how to go about it and identifies the places where creativity is needed. But you still have to supply the creativity...."[51]

Categorizing the systems development methods as empirical is critical to the effective management of the systems development process.

If systems development methods are categorized as empirical, measurements and controls are required because it is understood that the inner workings of the method are so loosely defined that they cannot be counted on to operate predictably.

In the past, methods have been provided and applied as though they were theoretical. As a consequence, measurements were not relied upon and controls dependent upon the measurements weren't used.

Many of the problems in developing systems have occurred because of this incorrect categorization. When a black box process is treated as a fully defined process, unpredictable results occur. Also, the controls are not in place to measure and respond to the unpredictability.

Chapter 4: Getting Started with Scrum

Scrum has three roles, three ceremonies, and three artifacts:

- Roles – Product Owner, ScrumMaster, Team
- Ceremonies – Sprint Planning, Daily Scrum, Sprint Review
- Artifacts – Product Backlog, Sprint Backlog, Burndown Chart

In order to get started with Scrum, everyone needs to understand roles and responsibilities as well as the operational flow of a Scrum. Units of work are broken down into discrete intervals call Sprints (sometimes called iterations) of 30 days or less. Practical experience has shown that teams cannot effectively maintain a global view of all the pieces of a project in more than 30 day chunks. All Agile processes use iterations that deliver working software that is "done" at the end of an iteration in the sense that features completed are potentially shippable to end users. This means the software must be tested to ensure that it works. Today, only about half the Scrum teams worldwide can produce tested software at the end of a Sprint. Extensive data from a CMMI Level 5 company shows that this failure mode tends to double defects and double development time.

To effectively deliver potentially shippable product at the end of the Sprint, experience has show that:

1. There must be one Product Owner that delivers the team a clear list of consistent priorities that are held fixed for the length of a Sprint. Decades of experience in software engineering have shown that giving a development team multiple priorities or changing priorities during an iteration causes delayed decisions and excessive rework. More than 50% reduction in productivity of such teams is the norm. Many development teams have doubled their productivity in the first month of Scrum implementation by fixing this problem.
2. The Product Owner must have a Product Backlog of features that are rank ordered by business value.
3. The work required to implement the features must be estimated by the team that will implement the features. Research has shown that while experts may accurately estimate how long it would take them to implement a feature, it is impossible for them to estimate the technical and domain knowledge of the delivery team, the competing priorities from other projects that impact team productivity, the human factors on a team such as motivation, conflict resolution, team spirit that vary over time, or the loss or addition of personnel to a team because of hiring, firing, vacations, or other factors that disrupt team stability.
4. The team must maintain a Burndown Chart and know how many features can be delivered at the end of each Sprint. The estimate of the number of features that can be delivered at the end of a Sprint is know as the team's "velocity." Sprints need to be the same fixed length so the Product Owner can use the velocity to build a release plan. Management and customers need to know the date of shipment of a new release. Without fixed length Sprints with known velocity, release dates are not predictable.
5. The team must be able to execute the Sprint without being disrupted by project leaders, management, or other chaos inducing forces in an organization.

While only about 10% of Scrum teams wordwide can meet these basic standards today, any description of how to get started with Scrum must address how to step by step move towards these basic Scrum practices.

Scrum on Large Projects: Distributed, Outsourced Scrum

First Scrum Scaling at IDX Systems 1996-2000

In 1996, Jeff Sutherland become SVP of Product Development at IDX Systems Corporation with a development team that grew to almost 600 people by 2000. Scrum was introduced immediately for all developers. A new organizational structure was created to coordinate Scrum teams called a Scrum of Scrums. Previous management positions were eliminated and all managers became team leaders. Typically, Directors of Engineering became Scrum of Scrum leaders. Ken Schwaber provided consulting experience in various parts of the development organization during 1996-2000 to help introduce Scrum to the organization.

Business units with product portfolios typically had less than 100 developers. A Scrum of Scrums was able to manage this size group effectively. At this level a lead architect became the Product Owner of the architecture backlog for the business unit and was the single person responsible for the architecture for that unit.

At a SVP level, there was a team of Directors and VPs that met periodically to coordinate activities across business units. The SVP, having s systems architecture background, led a virtual team of all the business unit architects to develop a global architecture backlog for all business units. Sprint planning involved the entire business unit once a month and the business unit architect was responsible for getting a commitment from each Scrum team to devote 10% of the resources in every Scrum team to addressing the global architecture backlog. This drove all business units incrementally towards a common architectural framework.

A similar virtual team strategy was used for software integration across business units and for common quality assurance processes and procedures. The virtual team strategy allowed all senior developers to be working with a Scrum team during every Sprint. The approach was designed to avoid separate specialized teams and to get everyone into front line production.

The SVP team worked well to coordinate Scrum of Scrums teams across business units. However, it was not optimal for driving a global Product Backlog across business units. Today, best practices are to implement a MetaScrum above the Scrum of Scrums teams. The MetaScrum is lead by the Chief Product Owner and incorporates all stakeholders in the company. Product release strategy and the state of every Sprint is reviewed at MetaScrum meetings. All decisions to start, stop, or change Sprints are made there. Often, the CEO is the ScrumMaster for the MetaScrum.

Linear Scalability in Large, Distributed, and Outsourced Scrums

Two case studies published in 2007 demonstrate for the first time that a software development process can scale linearly across both development team size and geographies. The SirsiDynix/StarSoft Development Labs project delivered over a million lines of code with teams distributed across the U.S., Canada, and Russia [18]. When the development team doubled in size by bringing on engineers in St. Petersburg, Russia, the velocity of software delivery more than doubled.

A similar effect was noted by a CMMI Level 5 Scrum implementation at Systematic Software Engineering in Denmark [19]. Introducing Scrum increased productivity is small teams only slightly as they were already Agile. On larger projects, they consistently achieved the same level of productivity per developer as on small teams. After achieving at 80% reduction in planning costs, a 50% reduction in total project costs, along with significantly increased user and employee satisfaction, they converted their company-wide CMMI Level 5 process documentation, training, and implementation to Scrum.

At Agile 2008, Systematic Software Engineering published a more detailed paper on combining CMMI with Scrum to produce a more disciplined environment for large projects while achieving the high velocity seen on their small Scrum projects. This paper is included and shows how Systematic has institutionalized a company wide CMMI Level 5 Scrum process.

There are many moving parts in an enterprise wide implementation of Scrum. While a clear, consistent model is achievable for any company, it must be localized into a specific company structure through inspection and adaptation, the hallmark of Scrum. This immediately leads to questions on how to best organize the company to take advantage of Scrum. It is strongly recommended to use expertise from an experienced Scrum Trainer who has led multiple enterprise Scrum implementations to work through best strategies for implementing enterprise Scrum in a specific company.

Agile Can Scale: Inventing and Reinventing Scrum in Five Companies

Jeff Sutherland, Ph.D.
PatientKeeper, Inc., 2001

Introduction

Agile development is focused on delivering maximum business value in the shortest possible time. It is well known that over half the requirements on a typical software project change during development and about half the features in delivered software are never used by customers. Adaptive planning and self-organizing teams are used to embrace changing requirements and avoid building features with low return on investment. The result is faster delivery, better quality, higher user satisfaction, and a more creative and enjoyable working environment for developers.

Scrum derives from Japanese best practices in lean manufacturing and lean product development [1, 13]. The goal of Scrum is to achieve the "Toyota Effect", deliver four times as much software with twelve times the quality within a series of short time boxes called "Sprint's," which last 30 days or less. Scrum is characterized by 15 minute, intensive, daily meetings of a multidisciplinary software delivery team, usually including product marketing, testers, software analysts, designers, and coders, and even deployment support staff. Prior to each iteration a list of features to be delivered called the Product Backlog is reprioritized so that maximum value features are developed first. Most tasks required to complete Product Backlog features are defined and estimated at the start of the Sprint by the multidisciplinary development team to product the Sprint Backlog. Some tasks require further design or definition during a Sprint so the Sprint Backlog will dynamically evolve, yet a skilled team will estimate the amount of work required to complete a Sprint with less than 20% error using improve estimating techniques based on the latest research on planning [30].

A lean production environment is based on a "pull" system [8] where team members pull inventory into a work environment "just in time." A the start of a Sprint the team pulls the amount of Product Backlog into the Sprint that they can commit to complete during the Sprint. Loading of team members during the Sprint is dynamic, in the sense that team members chose their own tasks and "pull" from the Sprint Backlog when they are ready to start another task. Good teams "minimize the work in progress" by keeping as few tasks open as possible to avoid integration problems and missed dates at the end of a Sprint.

Within a Sprint, problems and challenges are evaluated in the daily Scrum meeting and the team self organizes in real time to maximize the number of production ready features delivered by the team at the end of each Sprint. The daily meetings focused on answers to three questions by each member of the team – What did I do yesterday? What will I do today? What blocks, problems, or impediments are getting in my way? A well functioning team will dynamically recalculate the plan daily through a brief discussion and provide enough information for a ScrumMaster, the team leader, to calculate a "Burndown Chart" of work to be completed. Team efforts to

accelerate or decelerate the downward velocity of the burndown graph allow a team to "fly" the project into a fixed delivery date.

A typical Burndown Chart is illustrated below. It consists of the cumulative time it takes to complete outstanding tasks for deliverable software for a Scrum sprint. Each developer breaks down tasks into small pieces and enters into a backlog system work remaining on each active task every day. This could be as simple as task updates on a white board or an Excel spreadsheet, or could be a specialized tool for support of distributed teams. The remaining work for each task is added up to generate the cumulative backlog. Best current methods require only one minute of each developers time each day to update two data items for active tasks (percent complete and time remaining) allowing an automated system to produce the Burndown Chart, showing how fast outstanding backlog is decreasing each day. In the daily Scrum meetings, the team members use information sharing to determine what actions taken that day will maximize the download movement of the cumulative backlog. Experience has shown that Scrum project planning will consistently produce a faster path to the end goal than any other form of project planning reported to date, with less administrative overhead than any previously reported approach.

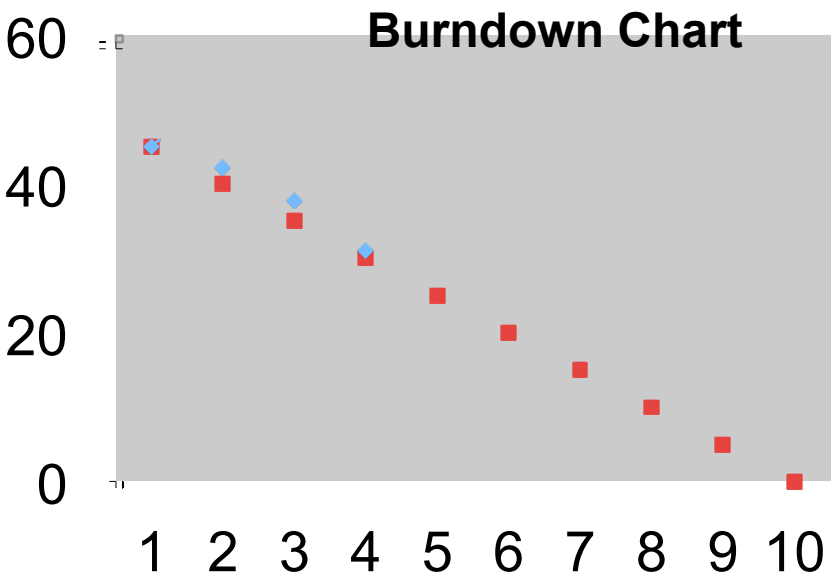


Figure 10: Burndown chart

Details of the Scrum approach have been carefully documented elsewhere [22]. Scrum is the only Agile methodology that has been formalized and published as an organizational pattern for software development [25]. The process assumes that requirements will change during the period between initial specification and delivery of a product. It supports Humphrey's Requirements Uncertainty Principle [52], which states that for a new software system, the requirements will not be completely know until after the users have used it. Scrum allows for Ziv's Uncertainty Principle in software engineering, which observes that uncertainty is inherent and inevitable in software development processes and products [53]. And it accounts for Wegner's mathematical proof (lemma) that it is not possible to completely specify an interactive system [54]. Most

software systems built today are object-oriented implementations, and most of those object-oriented systems depend on environmental inputs to determine process outputs (i.e., they are interactive systems).

Traditional, heavyweight software methodologies assume that requirements can be specified in advance, that they will not change during development, that the users know what they want before they see it, and that software development is a predictable, repeatable process. These assumptions are fundamentally flawed and inconsistent with the mathematical lemmas and principles cited above. As a result, 31% of software projects, usually driven by a variant of the waterfall methodology, are terminated before completion [55].

This article serves as a short retrospective on the origins of Scrum, its evolution in five companies, and a few key learnings along the way. It will provide a reference point for further investigation and implementation of Scrum for those interested in using a proven, scalable, lightweight development process that supports the principles of the Agile Alliance as outlined in the "Manifesto for Agile Software Development" (see www.agilealliance.org).

Easel Corporation: The First Scrum

Scrum was started in 1993 for software teams at Easel Corporation, where I was VP of object technology. In the initial Scrum, we built the first object-oriented design and analysis tool that incorporated round-trip engineering. The second Scrum implemented the first product to completely automate object-relational mapping in enterprise development environment. I was assisted by two world-class developers--Jeff McKenna, now a Scrum and eXtreme Programming consultant, and John's Scumniotales, now a development leader for object-oriented design tools at Rational Corporation.

In 1995, Easel was acquired by VMARK. Scrum continued there until I joined Individual in 1996 as VP of engineering to develop Personal NewsPage (now www.office.com). I asked Ken Schwaber, CEO of Advanced Development Methodologies, to help me incorporate Scrum into Individual's development process. On the same year, I took Scrum to IDX Systems when I assumed the position of senior VP of engineering and product development and CTO. IDX, one of the largest US healthcare software companies, was the proving ground for multiple team Scrum implementations. At one point, almost 600 developers were working on dozens of products. In 2000, Scrum was introduced to PatientKeeper, a mobile/wireless healthcare platform where I became CTO. So I have experienced Scrum in five companies that varied widely in size. They were proving grounds for Scrum in all phases of company growth from startup, to initial IPO, to mid-sized, and then to a large company delivering enterprise systems to the marketplace.

"All-at-Once" Software Development

There were some key factors that influenced the introduction of Scrum at Easel Corporation. The book Wicked Problems, Righteous Solutions [31] by Peter DeGrace and Leslie Hulet Stahl reviewed the reasons why the waterfall approach to software development does not work for software development today. Requirements are not fully understood before the project begins.

The users know what they want only after they see an initial version of the software. Requirements change during the software construction process. And new tools and technologies make implementation strategies unpredictable. DeGrace and Stahl reviewed "All-at-Once" models of software development, which uniquely fit object-oriented implementation of software and help to resolve these challenges.

"All-at-Once" models of software development assume that the creation of software is done by simultaneously working on requirements, analysis, design, coding, and testing and then delivering the entire system all at once. The simplest All-at-Once model is a single super-programmer creating and delivering an application from beginning to end. All aspects of the development process reside in a single person's head. This is the fastest way to deliver a product that has good internal architectural consistency, and it is the "hacker" mode of implementation. The next level of approach to All-at-Once development is handcuffing two programmers together, as in the XP practice of pair programming [23]. Two developers deliver the entire system together. This is been shown to deliver better code (in terms of usability, maintainability, flexibility, and extendability) faster than work delivered by larger teams. The challenge is to achieve a similar productivity effect in the large with an entire team and then with teams of teams.

Our team based All-at-Once model was based on both 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 Hirotaka Takeuchi and Hujiro Nonaka's description of the team-building process in setting up and managing a Scrum [1]. The idea of building a self-empowered team in which everyone had a global view of the product on a daily basis seemed like the right idea. This approach to managing the team, which had been so successful at Honda, Canon, and Fujitsu, resonated with the systems thinking approach being promoted by Peter Senge at MIT [15].

We were also impacted by recent publications in computer science. As I alluded above, Peter Wagner at Brown University demonstrated that it was impossible to fully specify or test an interactive system, which is designed to respond to external inputs (Wegner's lemma) [54]. Here was mathematical proof that any process that assumed known inputs, as does 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 James Coplien's paper on Borland's development of Quattro Pro for Windows [2]. The Quattro team delivered one million lines of C++ code in 31 months, with a four person staff growing to eight people later in the project. This was about a thousand 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.

Software Evolution and Punctuated Equilibrium

Our daily meetings at Easel were disciplined in a way that we now understand as the Scrum pattern [25]. The most interesting effect of Scrum on Easel's development environment was an observed "punctuated equilibrium effect." 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.

Evolutionary biologists have noticed that change occurs sharply at intervals separated by long periods of apparent stagnation, leading to the concept of punctuated equilibrium [6]. 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 [10]. 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 see how we could accelerate each other's work. For instance, one developer commented that if he changed a few lines in code, he could eliminate days of work for another developer. This effect was so dramatic that the project accelerated the point where it had to be slowed down. This hyperproductive state was seen in several subsequent Scrum's, but never went so dramatic as the one at Easel.

Stimulating Software Evolution with SyncSteps

The first Scrum worked from a unique view of the software system. 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 (see Figure 2). Topic areas are business object components. Changes are introduced into the system by introducing a unit of work that alters a component. Refactoring often causes a single change to ripple throughout the system. This unit of work in the initial Scrum was called a SynchStep.

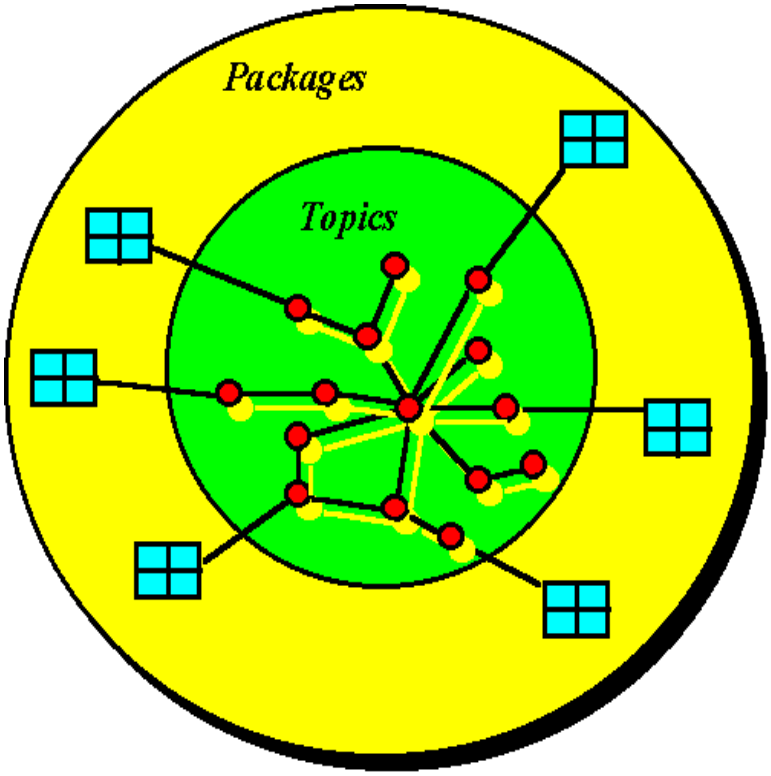


Figure 2: View of a software systems as seen by the first Scrum

System evolution proceeds in SyncSteps (see Figure 3). After one or more SyncSteps have gone to completion and forced some refactoring throughout the system, a new package of functionality emerges that is observable to the user. These SyncSteps are similar to genetic mutations. Typically, several interrelated components must mutate in concert to produce a significant new piece of functionality. This new functionality appears as the punctuated equilibrium effect to builders in the system. For a period of time, the system is stable with no new behavior. Then when a certain (somewhat unpredictable) SyncStep completes, the whole system pops up to a new level of functionality, often surprising the development team.

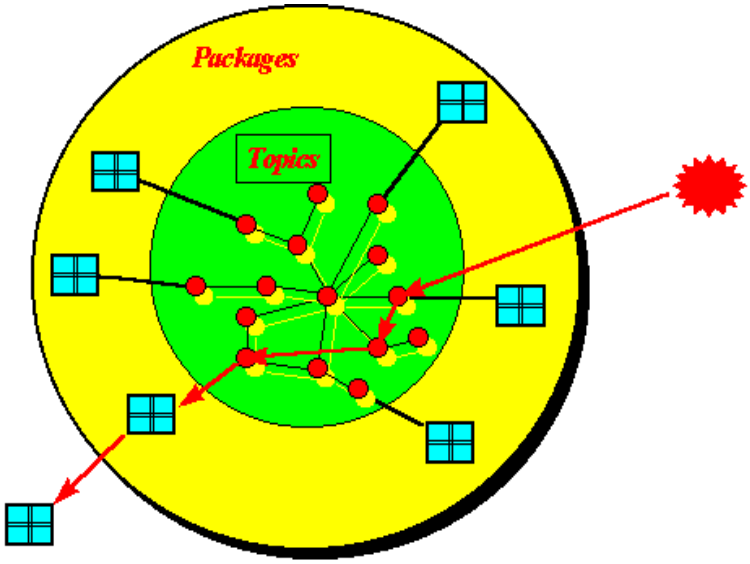


Figure 3: Firing a SyncStep causes a ripple through the system that triggers emission of a package of functionality visible to the user.

This aspect of self-organization is now understood as a type of Set-Based Concurrent Engineering (SBCE) which is practiced at Toyota [56]. Developers consider sets of possible solutions and gradually narrow the set of possibilities to converge on a final solution. Here, decisions on how and where to implement a feature in a set of components was delayed until the last possible moment. The most evolved component is selected “just in time” to absorb new functionality, resulting in minimal coding and a more elegant architecture. Thus emergent architecture, a core principle in all Agile processes, is not random evolution. Properly implemented, it is an SBCE technique viewed as a best business practice in some of the world’s leading corporations.

Achieving a Sustainable HyperProductive State

The key to entering a hyperproductive state was not just the Scrum organizational pattern. It was a combination of (1) the skill of the team, (2) the flexibility of a Smalltalk development environment, (3) the implementation of what are now know as XP engineering practices, and (4) the way we systematically stimulated production prototypes that rapidly evolved into a deliverable product.

Furthermore, in the hyperproductive state, the initial Scrum entered what professional athletes and martial artists call "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 was reminiscent of the Celtics basketball team at their peak, when they could do no wrong. The impact of entering the zone was not just hyperproductivity. Peoples personal lives were changed. Team members said they would never forget working on the project, and they would

always be looking for another experience like it. It induced open, team oriented, fun-loving behavior in unexpected persons. Those individuals who could not function well in an open, hyperproductive environment self-selected themselves out of the team by finding other jobs. This reinforced positive team behavior similar to biological systems, which select for fitness to the environment, resulting in improved performance of individual organisms.

VMARK: The First Senior Management Scrum

When Easel Corporation was acquired by VMARK (subsequenctly Informix, now Ascension Software) the original Scrum team continued its work on the same product. The VMARK senior management team was intrigued by Scrum and asked me to run a weekly senior management team Scrum to drive all the company's 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 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.

It was also at VMARK that Ken Schwaber was introduced to Scrum. Ken and I had worked together on and off for years. I showed him Scrum and he agreed it worked better that other project management approaches and was similar to how he built project management software in his company. He quickly sold off the project management software business and worked on bringing Scrum to the software industry at large. His work has had an incredible effect on deploying Scrum worldwide.

Individual: The First Internet Scrum

In the spring of 1996, I returned to Individual, Inc., a company I co-founded as VP of Engineering in 1988. Much of the Scrum experience at Individual has been documented by Ken Schwaber [17]. The most impressive thing to me about Scrum at Individual was not that the team delivered two new Internet products – and multiple releases of one of the products – in a single quarter. It was the fact that Scrum eliminated several hours a day of senior management meeting time starting the day that Scrum began, within a week of my arrival at the company. Because Individual 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. The management team was meeting daily to determine status of priorities that were viewed differently by every manager. These meetings were eliminated and the Scrum meetings became the focus for all decisionmaking.

It was incredibly productive to force all decisions to occur in the daily Scrum meeting. If anyone wanted to know the status of specific project deliverables or wanted to influence any priority, he or she could only do it in the daily Scrum meeting. I remember the senior VP 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 meetings. The Scrum reported status on the Web with green lights, yellow lights, and red lights for all pieces of functionality. In this way, the entire company knew status in real time, all the time. This transparency of information has become a key characteristic of Scum.

IDX Systems: The First Scrum in the Large

During the summer of 1996, IDX Systems hired me as senior VP of engineering and product development. IDX had over 4,000 customers and was one of the largest US 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 group into an interlocking set of Scrums. Every part of the organization was team based including the management team, which included two vice presidents, a senior architect, and several directors. Front-line Scrums met daily. A Scrum of Scrums, which included the team leaders of each Scrum in a product line, met weekly, The management Scrum met monthly.

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

One of the most productive teams at IDX was the Web Framerwork team that built a web frontend infrastructure for all products. The infrastructure was designed to host all IDX applications, as well as seamlessly interoperate with end user or third party applications. It was a distributed team with developers in Boston, Seattle, and Vermont who met by teleconference in a daily Scrum meeting. The geographic transparency of this model produced the same high performance as co-located teams and has become the signature of hyperproductive distributed/outsourced Scrums [28].

PatientKeeper Scrum: The First Scrum Company

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 back-end legacy systems. A robust 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. PatientKeeper proves that Scrum works equally well across technology implementations.

The key learning at PatientKeeper has involved the introduction of eXtreme Programming 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 XP. We were able to do some team programming and constant testing and refactoring, particularly as we migrated all development to Java and XML. It was more difficult to introduce these ideas when developers were working in C and C++. After a year of Scrum meetings in all areas of development, processes matured enough to capitalize on Scrum project management techniques, which were fully automated.

Complete automation and transparency of data allowed PatientKeeper to multithread Sprints through multiple teams. That in combination with implementing a MetaScrum of senior stakeholders in the company allowed PatientKeeper to run from top to bottom as a Scrum and become the first Scrum company to enter the hyperproductive state, delivering over 45 production releases a year of a large enterprise software platform. This became the prototype for the All-at-Once, or Type C Scrum, implemented in at least five companies by 2006 [40].

Conclusions

After introducing Scrum into five different companies of different sizes and with 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 organizational pattern with XP programming techniques, combined with innovative approaches to distributed teams and stimulation of rapid software system evolution. I believe these strategies can generate a hyperproductive Scrum on a predictable basis. The first Scrum did this intuitively and that was its key to extreme performance and a life-changing experience. In addition, the participation of Scrum leaders in the Agile Alliance [57], a group which has absorbed all leaders of well-known Agile development processes, will facilitated wider use of Scrum and its adoption as an enterprise standard development process.

Distributed Scrum: Agile Project Management with Outsourced Development Teams

Jeff Sutherland, Ph.D.
Patientkeeper
Newton, MA, US
jeff.sutherland@computer.org

Anton Viktorov
StarSoft Dev. Labs
St. Petersburg, Russia
anton.viktorov@starsoftlabs.com

Jack Blount
SirsiDynix
Provo, UT, USA
jack@dynix.com

Nikolai Puntikov
StarSoft Dev. Labs
St. Petersburg, Russia
nick@starsoftlabs.com

Abstract

Agile project management with Scrum derives from best business practices in companies like Fuji-Xerox, Honda, Canon, and Toyota. Toyota routinely achieves four times the productivity and 12 times the quality of competitors. Can Scrum do the same for globally distributed teams? Two Agile companies, SirsiDynix and StarSoft Development Laboratories achieved comparable performance developing a Java application with over 1,000,000 lines of code. During 2005, a distributed team of 56 Scrum developers working from Provo, Utah; Waterloo, Canada; and St. Petersburg, Russia, delivered 671,688 lines of production Java code. At 15.3 function points per developer/month, this is the most productive Java project ever documented. SirsiDynix best practices are similar to those observed on distributed Scrum teams at IDX Systems, radically different than those promoted by PMBOK, and counterintuitive to practices advocated by the Scrum Alliance. This paper analyzes and recommends best practices for globally distributed Agile teams.

Introduction

Scrum is an Agile software development process designed to add energy, focus, clarity, and transparency to project teams developing software systems. It leverages artificial life research [6] by allowing teams to operate close to the edge of chaos to foster rapid system evolution. It capitalizes on robot subsumption architectures [5] by enforcing a simple set of rules that allows rapid self-organization of software teams to produce systems with evolving architectures. A properly implemented Scrum was designed to increase speed of development, align individual and organization objectives, create a culture driven by performance, support shareholder value creation, achieve stable and consistent communication of performance at all levels, and enhance individual development and quality of life.

Scrum for software development teams began at Easel Corporation in 1993 and was used to build the first object-oriented design and analysis (OOAD) tool that incorporated round-trip engineering. In a Smalltalk development environment, code was auto-generated from a graphic design tool and changes to the code from the Smalltalk integrated development environment (IDE) were immediately reflected back into design.

A development process was needed to support enterprise teams where visualization of design immediately generated working code. This led to an extensive review of the computer science literature and dialogue with leaders of hundreds of software development projects. Key factors that influenced the introduction of Scrum at Easel Corporation were fundamental problems inherent in software development:

- Uncertainty is inherent and inevitable in software development processes and products - Ziv's Uncertainty Principle [53]
- For a new software system the requirements will not be completely known until after the users have used it - Humphrey's Requirements Uncertainty Principle [58]
- It is not possible to completely specify an interactive system – Wegner's Lemma [54]
- Ambiguous and changing requirements, combined with evolving tools and technologies make implementation strategies unpredictable.
- "All-at-Once" models of software development uniquely fit object-oriented implementation of software and help resolve these challenges. They assume the creation of software involves simultaneously work on requirements, analysis, design, coding, and testing, then delivering the entire system all at once [31].

"All-at-Once" Development Models

The simplest "All-at-Once" model is a single super-programmer creating and delivering an application from beginning to end. This can be the fastest way to deliver a product that has good internal architectural consistency and is the "hacker" model of implementation. For example, in a "skunk works" project prior to the first Scrum, a single individual surrounded by a support team spent two years writing every line of code for the Matisse object database [59] used to drive $10B nuclear reprocessing plants worldwide. At less than 50,000 lines of code, the nuclear engineers said it was the fastest and most reliable database ever benchmarked for nuclear plants. IBM documented a variant of this approach called the Surgical Team and considered it the most productive approach to software development [32]. The Surgical Team concept has a fatal flaw in that there are at most one or two individuals even in a large company that can execute this model. For example, it took three years for a competent team of developers to understand the conceptual elegance of the Matisse object server well enough to maintain it. The single-programmer model does not scale well to large projects.

The next level of "All-at-Once" development is handcuffing two programmers together. Pair programming, an eXtreme Programming practice [23], is an implementation of this. Here, two developers working at the same terminal deliver a component of the system together. This has been shown to deliver better code (usability, maintainability, flexibility, extendibility) faster than two developers working individually [33]. The challenge is to achieve a similar productivity effect with more than two people.

Scrum, a scalable, team-based "All-at-Once" model, was motivated by the Japanese approach to team-based new product development combined with simple rules to enhance team self-organization (see Brooks' subsumption architecture [5]). At Easel, the development team was already using an iterative and incremental approach to building software [34]. Features were

implemented in slices 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 for setting up and managing a Scrum [1]. The idea of building a self-empowered team in which a daily global view of the product caused the team to self-organize seemed like the right idea. This approach to managing the team, which had been so successful at Honda, Canon, and Fujitsu, also resonated with research on systems thinking by Professor Senge at MIT [15].

Hyperproductivity in Scrum

Scrum was designed to allow average developers to self-organize into high performance teams. The first Scrum achieved a hyperproductive state in 1993-1994 because of three primary factors. The first was the Scrum process itself, characterized by 15 minute daily meetings where each person answers three questions – what did you accomplish yesterday, what will you do today, and what impediments are getting in your way? This is now part of the standard Scrum organizational pattern [25]. Second, the team implemented all XP engineering processes [23] including pair programming, continuous builds, and aggressive refactoring. And third, the team systematically stimulated rapid evolution of the software system.

One of the most interesting complexity phenomena observed in the first Scrum was a "punctuated equilibrium" effect [35]. This phenomenon occurs in biological evolution when a species is stable for long periods of time and then undergoes a sudden jump in capability. Danny Hillis simulated this effect on an early super-computer, the Connection Machine [60].

"The artificial organisms in Hillis's particular world evolved not by steady progress of hill climbing but by the sudden leaps of punctuated equilibrium… With artificial organisms Hillis had the power to examine and analyze the genotype as easily as the realized phenotypes… While the population seemed to be resting during the periods of equilibrium … the underlying genetic makeup was actively evolving. The sudden increase in fitness was no more an instant occurrence than the appearance of a newborn indicates something springing out of nothing; the population seemed to be gestating its next jump. Specifically, the gene pool of the population contained a set of epistatic genes that could not be expressed unless all were present; otherwise the alleles for these genes would be recessive." [61]

Using Scrum with a fully integrated component design environment leads to unexpected, rapid evolution of a software system with emergent, adaptive properties resembling the process of punctuated equilibrium. Sudden leaps in functionality resulted in earlier than expected delivery of software in the first Scrum. Development tasks, originally planned to take days, could often be accomplished in hours using someone else's code as a starting point.

This aspect of self-organization is now understood as a type of Set-Based Concurrent Engineering (SBCE) practiced at Toyota [56]. Developers consider sets of possible solutions and gradually narrow the set of possibilities to converge on a final solution. Decisions on how and where to implement a feature is delayed until the last possible moment. The most evolved component is selected "just in time" to absorb new functionality, resulting in minimal coding and a more elegant architecture. Thus emergent architecture, a core principle in all Agile processes,

is not random evolution. Properly implemented, it is an SBCE technique viewed as a best business practice in some of the world's leading corporations.

Distributed, Outsourced Scrum

Scrum was designed to achieve a hyperproductive state where productivity increases by an order of magnitude over industry averages. Many small, collocated teams have achieved this effect. The question for this paper is whether a large, distributed, outsourced team can achieve the hyperproductive state.

U.S., European, or Japanese companies often outsource software development to Eastern Europe, Russia, or the Far East. Typically, remote teams operate independently and communication problems limit productivity. While there is a large amount of published research on project management, distributed development, and outsourcing strategies as isolated domains, there are few detailed studies of best project management practices on large systems that are both distributed and outsourced.

Current recommended Scrum practice is for local Scrum teams at all sites to synchronize once a day via a Scrum of Scrums meeting. Here we describe something rarely seen. At SirsiDynix, all Scrum teams consist of developers from multiple sites. While some Agile companies have created geographically transparency on a small scale, SirsiDynix uses fully integrated Scrum teams with over 50 developers in the U.S., Canada, and Russia. This strategy helped build a new implementation of platform and system architecture for a complex Integrated Library System (ILS). The ILS system is similar to a vertical market ERP system with a public portal interface used by more than 200 million people.

Best practices for distributed Scrum seen on this project consist of (1) daily Scrum team meetings of all developers from multiple sites, (2) daily meetings of Product Owner team (3) hourly automated builds from one central repository, (4) no distinction between developers at different sites on the same team, (5) and seamless integration of XP practices like pair programming with Scrum. While similar practices have been implemented on small distributed Scrum teams [39] this is the first documented project that demonstrates Scrum hyperproductivity for large distributed/outsourced teams building complex enterprise systems.

Distributed Team Models

Here we consider three distributed Scrum models commonly observed in practice:

- Isolated Scrums - Teams are isolated across geographies. In most cases off-shore teams are not cross-functional and may not be using the Scrum process.
- Distributed Scrum of Scrums – Scrum teams are isolated across geographies and integrated by a Scrum of Scrums that meets regularly across geographies.

- Totally Integrated Scrums – Scrum teams are cross-functional with members distributed across geographies. In the SirsiDynix case, the Scrum of Scrums was localized with all ScrumMasters in Utah.

Most outsourced development efforts use a degenerative form of the Isolated Scrums model where outsourced teams are not cross-functional and not Agile. Requirements may be created in the U.S. and developed in Dubai, or development may occur in Germany and quality assurance in India. Typically, cross-cultural communication problems are compounded by differences in work style in the primary organization vs. the outsourced group. In the worst case, outsourced teams are not using Scrum and their productivity is typical of waterfall projects further delayed by cross-continent communications lag time. Implementations of Scrum in a data rich CMMI Level 5 company simultaneously running waterfall, incremental, and iterative projects, showed productivity of Scrum teams was at least double that of waterfall teams, even with CMMI Level 5 reporting overhead [62]. Outsourced teams not using Scrum will typically achieve less than half the velocity of the primary site using Scrum.

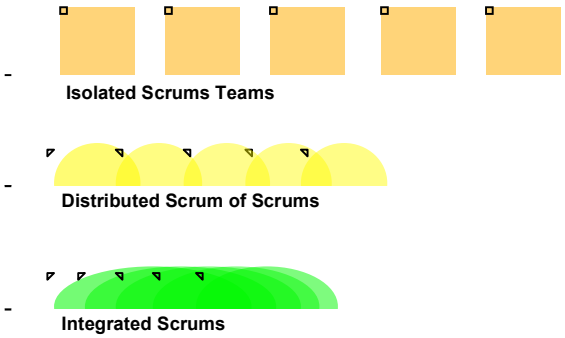


Figure 11: Strategies for distributed Scrum teams [11].

The latest thinking in the Project Management Institute Guide to the Project Management Body of Knowledge (PMBOK) models is a degenerative case of isolated non-Scrum teams [63]. This is a spiral waterfall methodology which layers the Unified Modeling Language (UML) and the Rational Unified Process (RUP) onto teams which are not cross-functional [64]. It partitions work across teams, creates teams with silos of expertise, and incorporates a phased approach laden with artifacts that violate the principles of lean development [12].

Best practice recommended by the Scrum Alliance is a Distributed Scrum of Scrums model. This model partitions work across cross-functional, isolated Scrum teams while eliminating most dependencies between teams. Scrum teams are linked by a Scrum-of-Scrums where ScrumMasters (team leaders/project managers) meet regularly across locations. This encourages communication, cooperation, and cross-fertilization and is appropriate for newcomers to Agile development.

An Integrated Scrums model has all teams fully distributed and each team has members at multiple locations. While this appears to create communication and coordination burdens, the daily Scrum meetings help to break down cultural barriers and disparities in work styles. On

large enterprise implementations, it can organize the project into a single whole with an integrated global code base. Proper implementation of this approach provides location transparency and performance characteristics similar to small co-located teams. A smaller, but similar, distributed team at IDX Systems Corporation during 1996-2000 achieved almost ten times industry average performance [39]. The SirsiDynix model approached this level of performance for distributed/outsourced Integrated Scrums. It appears to be the most productive distributed team ever documented for a large Java enterprise system with more than one million lines of code. This Integrated Scrums model is recommended for experienced Agile teams at multiple locations.

SirsiDynix Case Study

SirsiDynix has approximately 4,000 library and consortia clients, serving over 200 million people through over 20,000 library outlets in the Americas, Europe, Africa, the Middle East and Asia-Pacific. Jack Blount, President and CEO of Dynix and now CTO of the merged SirsiDynix company, negotiated an outsource agreement with StarSoft who staffed the project with over 20 qualified engineers in 60 days. Significant development milestones were completed in a few weeks and joint development projects are efficiently tracked and continue to be on schedule.

StarSoft Development Labs, Inc. is a software outsourcing service provider in Russia and Eastern Europe. Headquartered in Cambridge, Massachusetts, USA, StarSoft operates development centers in St. Petersburg, Russia and Dnepropetrovsk, Ukraine, employing over 450 professionals. StarSoft has experience handling development efforts varying in size and duration from just several engineers working for a few months to large-scale projects involving dozens of developers and spanning several years. StarSoft successfully uses Agile development and particularly XP engineering practices to maintain CMM Level 3 certification.

Hidden Costs of Outsourcing

The hidden costs of outsourcing are significant, beginning with startup costs. Barthelemy [65] surveyed 50 companies and found that 14% of outsourcing operations were failures. In the remainder, costs of transitioning to a new vendor often canceled out anticipated savings from low labor costs. The average time from evaluating outsourcing to beginning of vendor performance was 18 months for small projects. As a result, the MIT Sloan Management Review advises readers not to outsource critical IT functions.

The German Institute for Economic Research analyzed 43,000 German manufacturing firms from 1992-2000 and found that outsourcing services led to poor corporate performance, while outsourcing production helped [66]. While this is a manufacturing study rather than software development, it suggests that outsourcing core development may provide gains not seen otherwise.

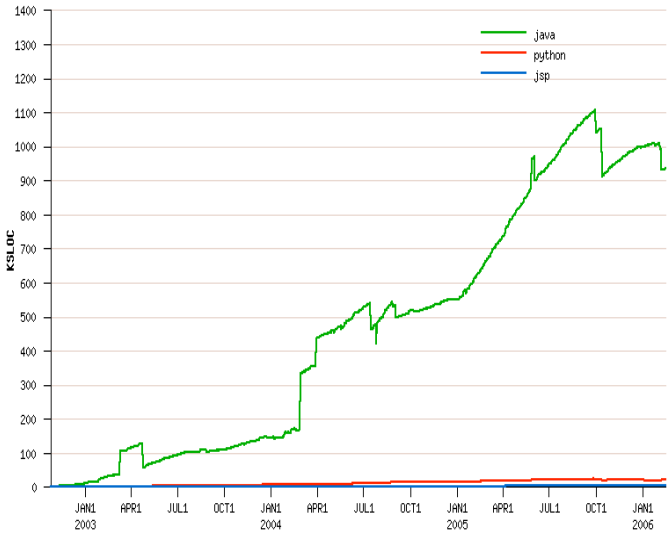


Figure 12 - SirsiDynix lines of new Java code in thousands from 2003-2006.

Large software projects are very high risk. The 2003 Standish Chaos Report show success rates of only 34%. 51% of projects are over budget or lacking critical functionality. 15% are total failures [67].

SirsiDynix sidestepped many of the hidden costs, directly outsourced primary production, and used Integrated Scrums to control risk. The goals of increasing output per team member and linearly increasing overall output by increasing team size were achieved. Production velocity more than doubled when the 30 person North American development team was augmented with 26 Russians from StarSoft in December 2005.

Intent of the Integrated Scrums Model

An Agile company building a large product and facing time-to-market pressure needs to quickly double or quadruple productivity within a constrained budget. The local talent pool is not sufficient to expand team size and salary costs are much higher than outsourced teams. On the other hand, outsourcing is only a solution if Agile practices are enhanced by capabilities of the outsourced teams. The primary driver is enhanced technical capability resulting in dramatically improved throughput of new application functionality. Cost savings are a secondary driver.

Context

Software complexity and demands for increased functionality are exponentially increasing in all industries. When an author of this paper flew F-4 aircraft in combat in 1967, 8% of pilot functions were supported by software. In 1982, the F16 software support was 45%, and by 2000, the F22 augmented 80% of pilot capabilities with software [63]. Demands for ease of use, scalability, reliability, and maintainability increase with complexity.

SirsiDynix was confronted with the requirement to completely re-implement a legacy library system with over 12,500 installed sites. Large teams working over many years in a changing business environment faced many new requirements in the middle of the project. To complicate matters further, the library software industry was in a consolidating phase. Dynix started the project in 2002 and merged with Sirsi in 2005 to form SirsiDynix.

Fortunately, Dynix started with a scalable Agile process that could adapt to changing requirements throughout the project. Time to market demanded more than doubling of output. That could only happen by augmenting resources with Agile teams. StarSoft was selected because of their history of successful XP implementations and their experience with systems level software.

The combination of high risk, large scale, changing market requirements, merger and acquisition business factors, and the SirsiDynix experience with Scrum combined with StarSoft success with XP led them to choose an Integrated Scrums implementation. Jack Blount's past experience with Agile development projects at US Data Authority, TeleComputing and JD Edwards where he had used Isolated Scrums and Distributed Scrum of Scrums models did not meet his expectations. This was a key factor in his decision to structure the project as Integrated Scrums.

Forces

Complexity Drivers

The Systems and Software Consortium (SSCI) has outlined drivers, constraints, and enablers that force organizations to invest in real-time project management information systems. Scalable Scrum implementations with minimal tooling are one of the best real-time information generators in the software industry.

SSCI complexity drivers are described as [63]:

- Increasing problem complexity shifting focus from requirements to objective capabilities that must be met by larger teams and strategic partnerships.
- Increasing solution complexity which shifts attention from platform architectures to enterprise architectures and fully integrated systems.
- Increasing technical complexity from integrating stand alone systems to integrating across layers and stacks of communications and network architectures.
- Increasing compliance complexity shifting from proprietary to open standards.
- Increasing team complexity shifting from a single implementer to strategic teaming and mergers and acquisitions.

SirsiDynix faced all of these issues. Legacy products were difficult to sell to new customers. They needed a new product with complete functionality for the library enterprise based on new

technologies that were highly scalable, easily expandable, and used the latest computer and library standards,

The SirsiDynix Horizon 8.0 architecture supports a wide variety of users from publication acquisition to cataloging, searching, reserving, circulating, or integrating information from local and external resources. The decision was made to use Java with J2EE, a modular design, database independency, maximum use of free platforms and tools, and wide support of MARC21, UNIMARC, Z39.50 and other ILS standards.

The project uses a three-tier architecture and Hibernate as a database abstraction layer. Oracle 10g, MS SQL, and IBM DB2 support is provided. The JBoss 4 Application server is used with a Java GUI Client with WebStart bootstrap. It is a cross-platform product supporting MS Windows 2000, XP, 2003, Red Hat Linux, and Sun Solaris. Built-in, multi-language support has on-the-fly resource editing for ease of localization. Other key technologies are JAAS, LDAP, SSL, Velocity, Xdoclet, JAXB, JUnit, and Jython.

Top Issues in Distributed Development

The SSCI has carefully researched top issues in distributed development [63], all of which had to be handled by SirsiDynix and StarSoft.

- **Strategic:** Difficult leveraging available resources, best practices are often deemed proprietary, are time consuming and difficult to maintain.
- **Project and process management:** Difficulty synchronizing work between distributed sites.
- **Communication:** Lack of effective communication mechanisms.
- **Cultural:** Conflicting behaviors, processes, and technologies.
- **Technical:** Incompatible data formats, schemas, and standards.
- **Security:** Ensuring electronic transmission confidentiality and privacy.

The unique way in which SirsiDynix and StarSoft implemented an Integrated Scrums model carefully addressed all of these issues.

Solution: Integrated Scrums

There are three roles in a Scrum: the Product Owner, the ScrumMaster, and the Team. SirsiDynix used these roles to solve the strategic distribution problem of building a high velocity, real-time reporting organization with an open source process that is easy to implement and low-overhead to maintain.

For large programs, a Chief ScrumMaster to run a Scrum of Scrums and a Chief Product Owner to centrally manage a single consolidated and prioritized product backlog is essential. SirsiDynix located the Scrum of Scrums and the Product Owner teams in Utah.

Team Formation

The second major challenge for large projects is process management, particularly synchronizing work between sites. This was achieved by splitting teams across sites and fine tuning daily Scrum meetings.

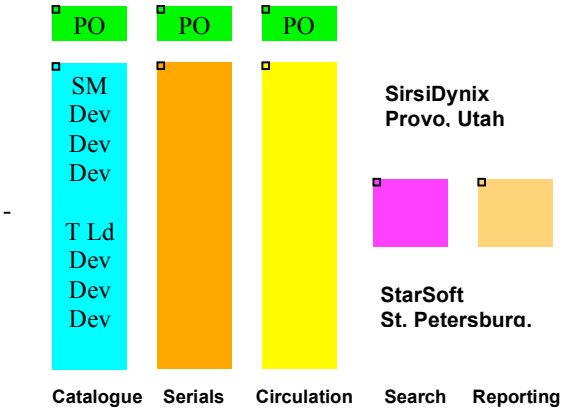


Figure 3 Scrum teams split across sites. PO=Product Owner, SM=ScrumMaster, TLd=Technical Lead.

Teams at SirsiDynix were split across the functional areas needed for an integrated library system. Half of a Scrum team is typically in Provo, Utah, and the other half in St. Petersburg. There are usually 3-5 people on the Utah part of the team and 4 or more on the St. Petersburg portion of the team. The Search and Reporting Teams are smaller. There are smaller numbers of team members in Seattle, Denver, St. Louis, and Waterloo, Canada.

Scrum Meetings

Teams meet across geographies at 7:45am Utah time which is 17:45 St. Petersburg time. Teams found it necessary to distribute answers to the three Scrum questions in writing before the Scrum meeting. This shortens the time needed for the join meeting teleconference and helps overcome any language barriers. Each individual reports on what they did since the last meeting, what they intend to do next, and what impediments are blocking their progress.

Email exchange on the three questions before the daily Scrum teleconference was used throughout the project to enable phone meetings to proceed more smoothly and efficiently. These daily team calls helped the people in Russia and the U.S. learn to understand each other. In contrast, most outsourced development projects do not hold formal daily calls and the communication bridge is never formed.

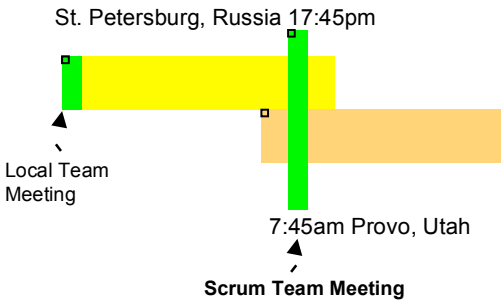


Figure 4 – Scrum Team meetings

Local sub-teams have an additional standup meeting at the beginning of the day in St. Petersburg. Everyone uses the same process and technologies and daily meetings coordinate activities within the teams.

ScrumMasters are all in Provo, Utah or Waterloo, Canada, and meet in a Scrum of Scrums every Monday morning. Here work is coordinated across teams. Architects are directly allocated to production Scrum teams and all located in Utah. An Architecture group also meets on Monday after the Scrum of Scrums meeting and controls the direction of the project architecture through the Scrum meetings. A Product Owner resident in Utah is assigned to each Scrum team. A chief Product Owner meets regularly with all Product Owners to assure coordination of requirements. SirsiDynix achieved strong central control of teams across geographies by centrally locating ScrumMasters, Product Owners, and Architects. This helped them get consistent performance across distributed teams.

Sprints

Sprints are two weeks long on the SirsiDynix project. There is a Sprint planning meeting similar to an XP release planning meeting in which requirements from User Stories are broken down into development tasks. Most tasks require a lot of questions from the Product Owners and some tasks take more time than initial estimates.

The lag time for Utah Product Owner response to questions on User Stories forces multitasking in St. Petersburg and this is not an ideal situation. Sometimes new tasks are discovered after querying Product Owners during the Sprint about feature details.

Code is feature complete and demoed at the end of each Sprint. Up until 2006, if it met the Product Owner's functional requirement, it was considered done, although full testing was not completed. It was not deliverable code until SirsiDynix strengthened its definition of "done" to include all testing in 2006. Allowing work in progress to cross Sprint boundaries introduces wait times and greater risk into the project. It violates the lean principle of reducing work in progress and increases rework.

Product Specifications

Requirements are in the form of User Stories used in many Scrum and XP implementations. Some of them are lengthy and detailed, others are not. A lot of questions result after receiving the document in St. Petersburg which are resolved by in daily Scrum meetings, instant messaging, or email.

Story for Simple Renewals Use Case:

Patron brings book or other item to staff to be renewed.

Patron John Smith checked out "The Da Vinci Code" the last time he was in the library. Today he is back in the library to pick up something else and brings "The Da Vinci Code" with him. He hands it to the staff user and asks for it to be renewed. The staff user simply scans the item barcode at checkout, and the system treats it as a renewal since the item is already checked out to John. This changes the loan period (extends the due date) for the length of the renewal loan. Item and patron circulation history are updated with a new row showing the renewal date and new due date. Counts display for the number of renewals used and remaining. The item is returned to Patron John Smith.

Assumptions:
Item being renewed is currently checked out to the active patron

- No requests or reservations outstanding
- Item was not overdue
- Item does not have a problem status (lost, etc)
- No renew maximums have been reached
- No block/circulation maximums have been reached
- Patron's subscriptions are active and not within renewal period
- No renewal charges apply
- No recalls apply

Renewal is from Check Out (not Check In)
Staff User has renewal privileges
Verification (How to verify completion):

- Launch Check Out
- Retrieve a patron who has an item already checked out but not yet overdue
- Enter barcode for checked out item into barcode entry area (as if it is being checked out), and press <cr>.
- System calculates new due date according to circulation rules and agency parameters.
- The renewal count is incremented (Staff renewal with item)
- If user views "Circulation Item Details", the appropriate Renewals information should be updated (renewals used/remaining)

- Cursor focus returns to barcode entry area, ready to receive next scan (if previous barcode is still displayed, it should be automatically replaced by whatever is entered next)
- A check of the item and patron circulation statistics screens show a new row for the renewal with the renewal date/time and the new due date.

For this project, St. Petersburg staff likes a detailed description because the system is a comprehensive and complex system designed for specialized librarians. As a result, there is a lot of knowledge that needs to be embedded in the product specification.

The ways libraries work in St. Petersburg are very different than English libraries. Russian libraries operate largely via manual operations. While processes look similar to English libraries on the surface, the underlying details are quite different. Therefore, user stories do not have sufficient detail for Russian programmers.

Testing

Developers write unit tests. The Test team and Product Owners do manual testing. An Automation Test team in Utah creates scripts for an automated testing tool. Stress testing is as needed.

During the Sprint, the Product Owner tests features that are in the Sprint backlog. Up until 2006, testers received a stable Sprint build only after the Sprint demo. The reason for this was a lower tester/developer ratio than recommended by the Scrum Alliance.
There are 30 team members in North America and 26 team members in St. Petersburg on this project. The St. Petersburg team has one project leader, 3 technical team leaders, 18 developers, 1 test lead, and 3 testers. This low tester/developer ratio initially made it impossible to have a fully tested package of code at the end of the Sprints.

The test-first approach was initially encouraged and not mandated. Tests were written simultaneously with code most of the time. GUIs were not unit tested.

Component	*Test Cases*	*Tested*
Acquisitions	529	384
Binding	802	646
Cataloging	3101	1115
Circulation	3570	1089
Common	0	0
ERM	0	0
Pac Searching	1056	167
Serials	2735	1714
Sub Total	11793	5115

Figure 5 – Test Cases Created vs. Tested (coded and working)

In the summer of 2006, a new CTO of SirsiDynix, Talin Bingham, took over the project and introduced Test Driven Design. Every Sprint starts with the usual Sprint Planning meeting and teams are responsible for writing functional tests before doing any coding. Once functional tests are written and reviewed, coding starts. Test-first coding is mandated. When coding is complete, developers run unit tests and manually pass all the functional tests before checking in changes to the repository.

Functional Area	Reserve Book Room
Task Description	Check that items from Item List is placed under Reserve with "Inactive" status
Condition	1. User has right for placing Items under Reserve 2. At least one Item List exists in the system 3. Default Reserve Item Status in Session Defaults is set to "Inactive"
Entry Point	Launcher is opened
Test Data	No specific data
Action	1. Reserve > Reserve Item 2. Select "Item Search" icon 3. Select "Item List" in the Combo box list of search options and enter appropriate Item list name 4. Press Enter 5. Select all Items which appear in the Item Search combo box and press "OK"
Expected Results	1. Items that were in Item list should appear in the list in Reserve Item 2. Status of all items that has been just added should be shown as "Inactive" 3. Save button should be inactive 4. All corresponding Item should retain their original parameters

Figure 6 – Functional Test Example

Automation testing is done using the Compuware TestPartner tool, but there is still room for improvement of test coverage.

Configuration Management

SirsiDynix was using CVS as source code repository when the decision was made to engage an outsourcing firm. At that time, SirsiDynix made a decision that CVS could not be used effectively because of lack of support for distributed development, largely seen in long code synchronization times. Other tools were evaluated and Perforce was chosen as the best solution.

StarSoft had seen positive results on many projects using Perforce. It is fast, reliable and offers local proxy servers for distributed teams. Although not a cheap solution, it has been very effective for the SirsiDynix project.

Automated builds run every hour with email generated back to developers. It takes 12 minutes to do a build, 30 minutes if the database changes. StarSoft would like to see faster builds and true concurrent engineering. Right now builds are only stable every two weeks at Sprint boundaries.

Pair Programming, Refactoring, and other XP practices

StarSoft is an XP company and tries to introduce XP practices into all their projects. Pair programming is done on more complicated pieces of functionality. Refactoring was planned for future Sprints and not done in every iteration as in XP. Some radical refactoring without loss of functionality occurred as the project approached completion. Continuous integration is implemented as hourly builds. On this project, these three engineering practices were used with Scrum as the primary project management methodology.

Measuring Progress

The project uses the Jira <http://www.atlassian.com> issue tracking and project management software. This gives everyone on the project a real-time view into the state of Sprints. It also provides comprehensive management reporting tools. The Figure below shows the Sprint burn-down chart, a snapshot of Earned Business, and a synopsis of bug status.

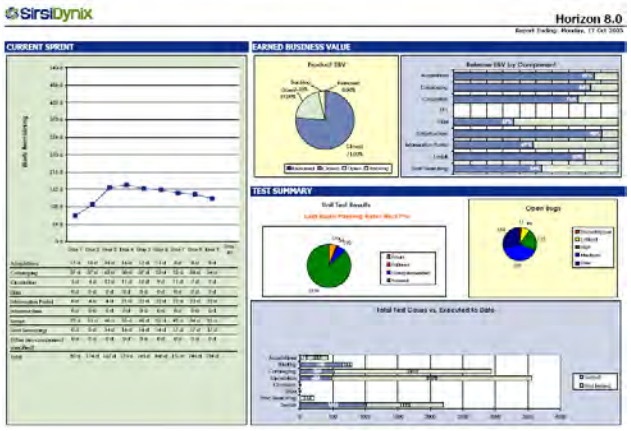


Figure 6 – SirsiDynix Horizon 8.0 Project Dashboard

Data from Jira can be downloaded into Excel to create any requested data analysis. High velocity projects need an automated tool to track status across teams and geographies. The best tools support bug tracking and status of development tasks in one system and avoid extra work on data entry by developers. Such tools should track tasks completed by developers and work remaining. They provide more detailed and useful data than time sheets, which should be avoided. Time

sheets are extra overhead that do not provide useful information on the state of the project, and are de-motivating to developers.

Other companies like PatientKeeper [68] have found tools that incorporate both development tasks and defects that can be packaged into a Sprint Backlog are highly useful for complex development projects. Thousands of tasks and dozens of Sprints can be easily maintained and reviewed real-time with the right tool.

Resulting Context with Integrated Scrums

Collaboration of SirsiDynix and StarSoft turned the Horizon 8.0 project into one of the most productive Scrum projects ever documented. For example, data is provide in the table below on a project that was done initially with a waterfall team and then re-implemented with a Scrum team [69]. The waterfall team took 9 months with 60 people and generated 54000 lines of code. It was re-implemented by a Scrum team of 4.5 people in 12 months. The resulting 50,803 lines of code had more functionality and higher quality.

	Scrum	Waterfall	SirsiDynix
Person Months	54	540	827
Java LOC	50,803	54000	671,688
Function Points	959	900	12673
FP per dev/month	17.8	2.0	15.3

Figure 7 – Function Points/Developer Month for collocated vs. distributed projects.

Capers Jones of Software Productivity Research has published extensive tables on average number of function points per lines of code for all major languages [70]. Since the average lines of code per function point for Java is 53, we can estimate the number of function points in the Scrum application. The waterfall implementation is known to have fewer function points. Distributed teams working on Horizon 8.0 generated 671,688 lines of code in 14.5 months with 56 people. During this period they radically refactored the code on two occasions and reduced the code based by 275,000. They have not been penalized for refactoring as that is rarely done in large waterfall projects in the database from which Capers derived his numbers. They have also not been rewarded for refactoring even though reducing lines of code is viewed as important as adding new code on well-run Agile projects.

Jones has also shown from his database of tens of thousands of projects that industry average productivity is 12.5 function points per developer/month for a project of 900 function points and that this drops to 3 for a project with 13000 function points [7]. Some of this is due to 4GL and other code-automation tools used on small projects, many of which are not implemented in third generation languages like Java.

The SirsiDynix project is almost as productive as the small Scrum project with a collocated team of 4.5 people. For a globally dispersed team, it is one of the most productive projects ever documented at a run rate of five times industry average.

Conclusions

This case study is a proof point for the argument that distributed teams and even outsourced teams can be as productive as a small collocated team. This requires excellent implementation of Scrum along with good engineering practices. The entire set of teams must function as a single team with one global build repository, one tracking and reporting tool, and daily meetings across geographies.

Outsourced teams must be highly skilled Agile teams and project implementation must enforce geographic transparency with cross-functional teams at remote sites fully integrated with cross-functional teams at the primary site. In the SirsiDynix case, the teams were all run from a central site giving strong central control.

It is highly unlikely that distributed outsourced teams using current Agile Alliance best practices of distributing work to independent Scrum teams across geographies could achieve the level of performance achieved in this case study. Therefore, SirsiDynix sets a new standard of best practices for distributed and outsourced teams with a previously demonstrated high level of Agile competence.

Fully Distributed Scrum: Replicating Local Productivity and Quality with Offshore Teams

Jeff Sutherland, Ph.D
Scrum, Inc.
Boston, MA, US
jeff.sutherland@scruminc.com

Guido Schoonheim
Xebia b.v.
Hilversum, Netherlands
gschoonheim@xebia.com

Maurits Rijk
Xebia b.v
Hilversum, Netherlands
mrijk@xebia.com

Abstract

Scrum was designed for hyperproductive teams where productivity increases by 5-10 times over industry averages and many colocated teams have achieved this effect. The question for this paper is whether distributed, offshored teams can consistently achieve the same level of performance. In particular, can a team establish a localized velocity and quality and then maintain or increase that velocity and quality when distributing teams across continents. Since 2006, Xebia (Netherlands) started localized projects with half Dutch and half Indian team members. After establishing localized hyperproductivity, they move the Indian members of the team to India and show the same velocity with fully distributed teams. After running XP engineering practices inside many distributed Scrum projects, Xebia has systematically productized a model for high performance, distributed, offshore teams with one of the lowest defect rates in the industry.

Introduction

The advantages of colocated teams, doubling developer productivity compared to non-colocated teams, are commonly eliminated by distributed software development and offshoring [1]. This paper introduces a model for producing distributed and offshored team velocity that is equal to colocated velocity of a single team. The model is repeatable, proven across many projects, and is recommended for teams that can execute a high performance Scrum implementation [2] with XP engineering practices inside [3].

Agile project management with Scrum derives from best business practices in companies like Fuji-Xerox, Honda, Canon, and Toyota [4]. Combining Scrum with XP engineering practices has generated hyperproductive teams with 5-10 times industry average performance since 1993 [5] [6]. In 2005, two Agile companies, SirsiDynix (U.S.) and Exigen Services (Russia), used distributed Scrum teams to deliver linearly scalable performance for a large project of over 1M lines of code [7]. A distributed team of over 50 people in the U.S. and Russia delivered velocity per developer equivalent to a 6 person hyperproductive, colocated Scrum team. This 50 person team produced the same number of features as a typical 350 person waterfall team [8].

During 2006-2008, Xebia implemented a distributed software development team model on multiple projects of variable types with teams half in the Netherlands and half in India. These distributed teams used the Scrum process with XP engineering practices inside. When replicated over multiple projects, the Xebia implementation shows distributed velocity equal to local velocity and has created a new recommended standard for deploying distributed teams across geographies.

Here we describe offshoring strategies for overcoming the typical geographic, language, and cultural barriers that impede distributed development. Traditional outsourcing failures can be avoided with the approach described here. Distribution of individual Scrum teams across geographies eliminates communication failures, XP practices solve integration problems, and daily team meetings maintain high focus on customer priorities. Earlier work in other companies showed that colocation doubled Agile team productivity [1]. The fully distributed model supports geographically transparent software development projects where performance consistently meets or exceeds productivity of colocated Agile teams.

Benefits and challenges in outsourcing offshore

U.S., European, or Japanese companies often outsource software development to Eastern Europe, Russia, or the Far East. The three key advantages that offshoring strives to achieve are (1) lower costs of labor, (2) capture talent not available locally, and (3) increase and decrease project size without layoffs. Each of these advantages comes with its own challenges that have to be solved to make outsourcing successful.

Lower cost of labor

Typically, remote teams operate independently and communication problems lower productivity. Most offshoring organizations require detailed specifications before they begin a project and theses traditional project planning methodologies show high failure rates.

The hidden costs of offshoring are significant, beginning with startup costs. Barthelemy [65] surveyed 50 companies and found that 14% of outsourcing to offshore operations were failures. In the remainder, costs of transitioning to a new vendor often canceled out anticipated savings from low labor costs. The average time from evaluating offshoring to beginning of vendor performance was 18 months for small projects. As a result, the MIT Sloan Management Review advises readers not to outsource critical IT functions offshore.

Secondly, high productivity is not easily achievable. IDX Systems (now GE Healthcare) averaged 240% productivity improvement with Scrum during 1996-2000 based on analysis by external function point experts. Outsourcing development to an offshored waterfall team typically saves 20% over internal waterfall costs. At IDX, it would cost twice as much to get a project done offshore compared to internal Scrum teams. At PatientKeeper (a MIT startup company in 2000) during 2004-2007, the break even point for outsourcing was achieved only when Indian developers cost less than 10% of American developers. Because Indian waterfall developers cost 30% of Boston Scrum developers, it cost three times as much to get software developed by an Indian waterfall team. The PatientKeeper Board permanently terminated outsourcing after reviewing these ROI data.

Capture talent

Capturing external talent may also be a problem. Jack Blount, CEO of Dynix and former COO of Borland [18] decided not to outsource to India and China after he verified that annual turnover rates were 30-50% [9]. Rathi describes how employee turnover is largely determined by two variables: person-culture and person-job fit [10]. Lack of these fits results in lower job satisfaction and consequently in employee turnover.

Scale project size without layoffs

Scaling with remote capacity gives you a local team that remains stable when you scale down. However if core development is moved offshore, knowledge gets lost when you downsize, causing severe problems and sometimes vendor lock-in.

Distributed Scrum team models

Achieving promised benefits of outsourcing requires real cost savings, stable offshore teams, and a strategy for retaining core knowledge onshore. This can be achieved with stable offshore Agile teams that can maintain the same velocity as onshore teams and with onshore teams that maintain the same knowledge level as offshore teams.

Here we consider three distributed Scrum models commonly observed in practice.

Isolated Scrums - Teams are isolated across geographies.

Distributed Scrum of Scrums – Scrum teams are isolated across geographies and integrated by a Scrum of Scrums that meets regularly across geographies.

Fully distributed Scrums – Scrum teams are cross-functional with members distributed across geographies. This means that a single team will have members in multiple locations. A single team might have a ScrumMaster and two developers in the Netherlands while a tester and four developers reside in India. These team members share a single sprint backlog and share code ownership. In the SirsiDynix case, the Scrum of Scrums was localized with all ScrumMasters in Utah. At Xebia, ScrumMasters may be in the Netherlands or India depending on project needs.

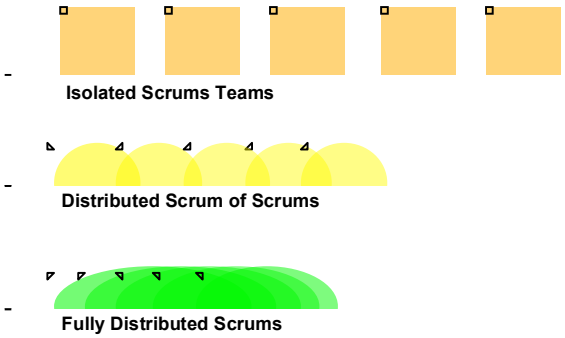


Figure 1: Distributed Scrum Team Strategies

Most offshore development efforts use a degenerative form of the Isolated Scrums model where outsourced teams are not cross-functional and not Agile. Requirements may be created in the U.S. and developed in Dubai, or development may occur in Germany and quality assurance in India. Typically, cross-cultural communication problems are compounded by differences in work style in the primary organization vs. the offshored group. In the worst case, teams outsourced this way are not using Scrum and their productivity is typical of waterfall projects further delayed by cross-continent communications lag time. Implementations of Scrum in a data rich CMMI Level 5 company simultaneously running waterfall, incremental, and iterative projects, showed productivity of Scrum teams was at least double that of waterfall teams, even with CMMI Level 5 reporting overhead [11]. Outsourced teams not using Scrum will in the best case achieve less than half the velocity of a onshore site using Scrum assuming equal talent across teams.

The latest thinking in the Project Management Institute Guide to the Project Management Body of Knowledge (PMBOK) models is a degenerative case of isolated non-Scrum teams . This is a spiral waterfall methodology which layers the Unified Modeling Language (UML) and the Rational Unified Process (RUP) onto teams which are not cross-functional [12]. It partitions work across teams, creates teams with silos of expertise, and incorporates a phased approach laden with artifacts that violate the principles of lean development [13].

Best practice recommended by the Scrum Alliance is a Distributed Scrum of Scrums model. This model partitions work across cross-functional, isolated Scrum teams while eliminating most dependencies between teams. Scrum teams are linked by a Scrum-of-Scrums where ScrumMasters (team leaders/project managers) meet regularly across locations. This encourages communication, cooperation, and cross-fertilization and may be appropriate for newcomers to Agile development or those who have offshore limitations that cripple the productivity of the fully distributed model.

OneTeam Model

Xebia's Fully Distributed Scrum model has all teams fully distributed and each team has members at multiple locations. While this "OneTeam" model might seem to create communication and coordination burdens, most communication is handled by following the Scrum cycle. The daily Scrum meetings actually help to break down cultural barriers and disparities in work styles while simultaneously enhancing customer focus and offshore understanding of customer needs. On enterprise implementations, it can organize the project into a single whole with an integrated global code base. Proper implementation of OneTeam provides location transparency and performance characteristics similar to hyperproductive co-located teams.

Maximum business value is delivered in Scrum by implementing the Product Backlog in order of business value of features. Xebia team product features are represented in user stories and size of a story is represented in story points [5].

Xebia teams consistently validate that distributed velocity equals colocated velocity by measuring cost per story point. The value of the feature divided by actual cost is the prime indicator of business value and this is directly proportional to the velocity of the team in story points per iteration. The Fully Distributed Scrums model is recommended for experienced Agile teams in multiple locations because cost per story point is the same as localized teams and the Xebia distributed teams have improved focus on executing stories that better fit customer needs in the right order than localized teams.

Cost per story point cannot be standardized across the industry. The best standard metric to compare productivity across projects is Function Points. Capers Jones demonstrated many years ago that the number of features delivered in Function Points can be estimated by "back-firing" using lines of code delivered [8]. While this is an indirect measure of business value delivered, it is the best measure available to compare teams industry wide.

While one might argue that delivering lots of code may not produce business value, this is controlled by Scrum teams running XP engineering practices in two ways:

- Scrum orders Product Backlog by business value and assures lines of code delivered directly increase business value.
- The XP practice of refactoring eliminates many thousands of lines of code that would remain static in the code base of a waterfall team.

The net result is that comparisons of business value delivered by Scrum/XP teams is conservative compared to waterfall teams when measured by any indicator affected by lines of code.

Thus the message of this paper is that Xebia Scrum/XP teams deliver Function Points over seven times faster than industry average waterfall teams and the Function Points they deliver have higher business value than the waterfall teams by over an order of magnitude. Since this value is delivered at the same cost per story point, and this cost is a direct indicator of business value, either locally or distributed, the OneTeam model is recommended for distributed development by those Agile teams capable of executing it.

Xebia ProRail PUB Case Study

The model for Fully Distributed Scrums is best illustrated by a real life example of a Xebia OneTeam project; the ProRail PUB project.

ProRail, the logistical and infrastructural part of the Dutch railways, has been developing a new information system for travelers. Information about train departure times is stored centrally and updated with information from the rail network. When a train is delayed or arrives early this information is captured by sensors in the infrastructure as well as by manual actions to update train information.

The publishing of this information to travelers on all the railway stations throughout the Netherlands is the scope of Xebia's development assignment. Development included the aggregation and distribution system (combining real time information about multiple trains into messages relevant for stations), the client in the displays, the audio system and the controlling and monitoring interfaces. As this is a mission critical, high-availability enterprise system with large visibility, the non-functional requirements are extensive.

Xebia took over this project from a failed waterfall implementation and meeting deadlines was now a key criteria. The transparency and empirical project control that Scrum delivers were key incentives for the client to engage Xebia. The choice to make it an offshore project was driven by cost and scalability.

While Scrum is simple to understand, it is not easy to implement and distributed development adds another layer of complexity. The PUB project encountered a number of challenges in these areas.

Lower cost of labor

The OneTeam approach for Fully Distributed Scrum teams delivers the same results as a well running colocated Scrum team in an offshoring situation. Different aspects of the PUB project can illustrate this.

Productivity.

Velocity of a Scrum team is determined by the number of story points that the team can complete using a standardized definition of "done" in a single iteration. As story points are not translatable between projects the PUB project size has also been measured in function points. This measure has been done for both the old (failed) implementation and the new implementation by Xebia and these figures correspond. As a measurement this does not give a completely accurate picture, as it does not capture the amount of business value delivered very well. It is however the best means available to make comparisons over projects. Below is a table taken from a colocated 6 person Scrum, the SirsiDynix fully distributed Scrum project, and extended with PUB data.

	Cohn Colocated Scrum	Cohn Waterfall	SirsiDynix Distributed Scrum	Xebia Distributed Scrum
Person Months	54	540	827	125
Lines of Java	51000	58000	671688	100000+
Function Points	959	900	12673	1887
FP per dev. per month	17.8	1.7	15.3	15.1

Table 1: Productivity of Colocated Scrum vs. Waterfall Team [5], SirsiDynix Distributed Scrum [9], and Xebia OneTeam.

As we can see the Scrum projects easily outperform the Waterfall project. Xebia Distributed Scrum comes close to the small colocated Scrum team. The performance of the SirsiDynix and Xebia project is very similar. This shows that the high performance fully distributed Scrum approach is reproducible and not typical for only the SirsiDynix environment.

To investigate the effect of distributing teams on the productivity we can look at the cost per story point delivered throughout the project.

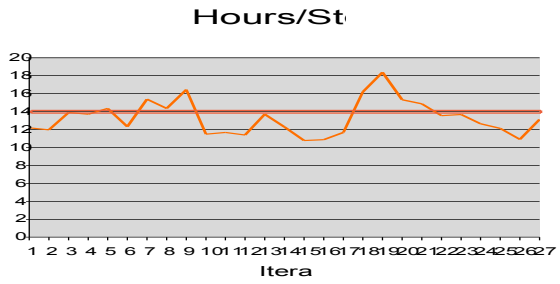


Figure 3: Hours per story point during the project

It is important to note the gradual increase in story point cost during the life of most projects due to growing complexity and growing codebase. This constant has been compensated for to focus the above diagram on any outliers. The transition from a local team to a distributed team took place at iteration 6. As can be seen from the resulting graph, the number of hours needed to implement a story point was not affected by this distribution. Storypoint estimates were determined at the beginning of the project for the whole product backlog and were determined for new requirements as they surfaced. Iteration 18 and 19 show a significant increase in hours needed per story point. Technical debt had been built up during the previous iterations. Starting with iteration 20 this technical debt was consistently removed, resulting in a gradual increase in productivity.

High quality and consistency.

Throughout the course of the PUB project a lot of attention has been paid to quality. The Scrum definition of done for this project includes unit test coverage of at least 80%, fully automated functional testing, full regression testing, performance and load testing for all implemented stories as well as updating the necessary documentation.

For every piece of functionality the whole team discusses proper design and necessary refactoring takes place. In addition to this shared ownership over design every team employs a 'quality watchdog'. This is a team member accountable for quality and consistency. Any problems that he / she signals are to be picked up and discussed by the team. All teams share the same team room and team members participate in design discussions of other teams in order to maintain architectural consistency across teams. Pair programming and rotation of people between teams is used to avoid code ownership and spread knowledge.

All issues that are found outside the iteration are measured and solved as shown in the following graph.

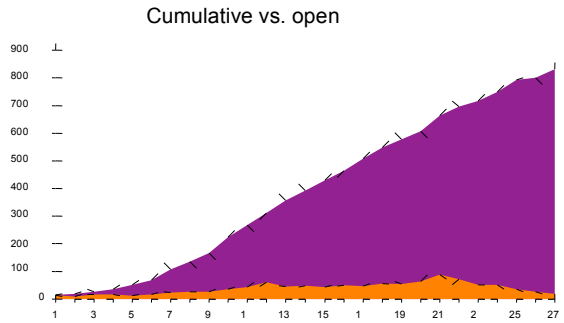


Figure 4: Open defects during the proect

The above figure shows that the numbef open defects remains constant (around 50). This means that the project is not building up technical debt during development. The number of open bugs per KLOC is actually decreasing because the code base is continuously growing. Other data also shows that more than 90 % of the defects found are solved in the same iteration in which they were introduced.

Based on these numbers we can conclude that the verification and validation process has isolated 6 defects per KLOC. During acceptance tests less than 1 defect per KLOC was found. A fair estimate is that 50% of the defects are still left in the product after the acceptance test, leaving us with 1 defect per KLOC. This is far less than industry average, which is around 5 defects per KLOC [14]. Fully Distributed OneTeam Scrums applying XP practices produce extremely high quality.

Capture talent not available locally

As described in 2.2 this benefit depends on the employee turnover. To cope with high turnover rates the project concentrated on clear communication, and special attention to the people-culture and people-job fit. In addition the Agile way of working provides talented people with responsible work, thus guaranteeing job satisfaction. This resulted in a turnover of less than 5% in one year.

Cultural differences.

Indian and Dutch team members have a different background and culture. This shows most clearly in communication. For example, where Dutch team members can be loud and direct, Indian team members can be careful and cautious in their expression. Also India is more hierarchically oriented than the Netherlands. The first and most important thing to counter these differences is good personal relationships. By traveling at the beginning and throughout the project, by seeing each other daily in video conference stand-ups, and by being part of the same team personal relationships formed. Secondly, a team culture aimed at openness and direct communication was actively developed by the ScrumMasters. This helped bring out issues during retrospectives and lowered communication barriers. Thirdly, a company culture of openness with an equal value system on both sites supported the team culture and made identifying with each other easier.

Sharing context and priorities

In an offshoring situation it is difficult to fully communicate all client nuances, context and priorities to offsite team members. To actively distribute this knowledge Xebia scheduled regular traveling, always-open Skype connections, a project news gazette after every iteration and informal updates by the product owner.

Clear communication through Scrum

The Scrum meetings facilitate practically all necessary communication. This is only possible because the team is fully distributed and shares the same sprint goals. All Scrum meetings were done in a distributed way using video conferencing via a simple Skype video call with the exception of the Demo. Separate meeting rooms are set up with conference equipment and a Scrum planning tool with a digital burn down chart is used to share the status of the sprint across locations. A microphone is passed around as ‘talking stick’ to facilitate clear audibility. Xebia found that face to face visuals greatly increases the effectiveness of communication and enhances personal relationships.

The Sprint planning meeting is done with the whole team using planning poker [15] so that members on both shores contribute to the estimation process. Planning a distributed sprint took 4 hours on average using two week sprints.

The daily standup meetings are done when the developers in the Netherlands come to work. A distributed standup lasts no longer then 15 minutes. A Scrum of Scrums meeting was held by ScrumMasters after the stand-ups to synchronize any dependant issues or impediments as well as technological issues.

The Scrum demo was not distributed in this case to provide maximum focus and responsiveness to the customer. The Dutch members briefed the Indian members after every Demo. The Scrum retrospective goes in the same fashion as the Sprint planning meeting. The distributed retrospective is completed in 2 hours.

Together these meetings provide the full official meeting cycle. One on one meetings are being held as necessary, as well as design discussions. This is no different from a colocated Scrum with the possible exception of tooling.

Some work is local

While all development work can be distributed there is project work that is not easily done in a distributed way. A fourth Scrum team, consisting only of local team members, was dedicated to specific customer facing compliancy activities and removing certain impediments. Examples of local deliverables are writing Dutch documentation, aligning with customer architectural stakeholders, discussing requirements with technical stakeholders and researching technical dependencies between the infrastructure and other systems. This resulted in clearing of a lot of roadblocks and a high velocity for the distributed teams.

Tooling for communication and process

In this project ScrumWorks [16] was used to manage the product backlog and sprint backlog electronically. Burndown graphs were printed everyday and posted on the wall in the team rooms.

For global sharing of information and documentation a wiki was used intensively. To discuss architecture a smartboard (computerized whiteboard) was used, along with other solutions for digital whiteboarding. A single code repository, single continuous build system, test servers accessible from both locations and a shared mailing list are some of the tools used to facilitate the development process.

Project structure and scaling

Xebia initiated the PUB project with a short initiation phase where the product backlog was developed, basic architecture constraints were established and processes such as QA, Acceptance and Requirements management were set up with the customer to match the client organization in an Agile way.

After three weeks of project initiation, a colocated development team started the first iteration of the project. Iteration length was set at two weeks throughout the project.

The first two iterations were done with Dutch developers. Indian team members were included onsite as soon as immigration and logistical constraints allowed, starting with the third iteration. Both Dutch and Indian team members worked as a single colocated Scrum team with a single sprint backlog, following all XP engineering practices. In the shared onsite iterations the team members forged personal relationships to last throughout the project. By being onsite with the customer, the Indian team members acquired a good sense of customer context. It also got everyone aligned concerning practices, standards, tooling, and natural roles in the team formed. After three iterations the onsite Indian team members returned to India. During these 5 initial iterations (10 weeks) the team established colocated hyperproductivity.

The project scaled up after Indian team members returned home. Engineers were added and two new full teams were formed. Engineers in the Netherlands were split over both teams as were engineers in India, creating two teams that both have members in multiple locations. Careful attention is paid to spreading the experience among the new teams and practices like pair programming are used to get new members up to speed. This cell division like process is repeated until the project is at the desired scale.

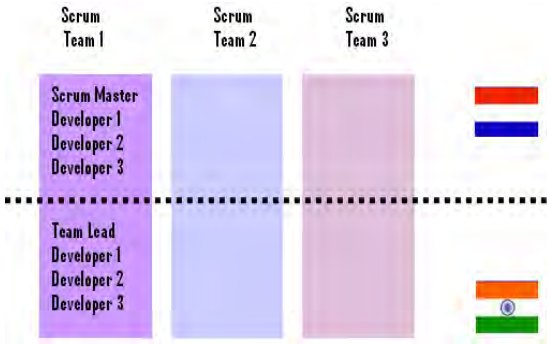


Figure 2: Fully Distributed Scrum team division

The project scaled up to three fully distributed Scrum teams and a fourth local Scrum team, with a total of 25 people. The different teams shared the same product backlog but used their own sprint backlogs.

At the end of the project the teams were scaled down and merged. As the client preferred to work with Dutch engineers for maintenance the Indian side was scaled down further. This was no problem since the use of distributed teams also ensures distributed knowledge.

The total size of the Xebia realization on this project is about 20 man-years, 100.000+ lines of code over a period of 11 months.

Conclusions

In summary, it is possible to create a distributed/outsourced Scrum with the same velocity and quality as a colocated team and this capability has been reproducible over many projects. The OneTeam strategy lower costs, captures offshore talent and allows increasing and decreasing team size without knowledge loss. We highly recommend this strategy for experienced Agile teams.

Future Research

Several other Xebia projects achieved the same velocity and quality as the PUB project confirming the OneTeam capability of distributing localized velocity and quality across continents. However, during the PUB project data collection was standardized and refined to a high level. Future projects will use the same data collection standards as the PUB project allowing a larger prospective study of the OneTeam effect with comparability across many projects.

While Xebia has now repeatedly demonstrated the effectiveness of the fully distributed team model, it requires Agile teams to fully implement the practices of Scrum and XP. Less than 10% of Agile teams worldwide are capable of doing this in 2008. Additional studies of the fully distributed model across multiple companies would be useful and may help some companies move beyond partial implementation of Agile practices in order to achieve the fully distributed model benefits.

References

[1] S. Teasley, L. Covi, M. S. Krishnan, and J. S. Olson, "How Does Radical Collocation Help a Team Succeed?," in *CSCW'00* Philadelphia, PA: ACM, 2000, pp. 339-346.

[2] J. Sutherland, "Future of Scrum: Parallel Pipelining of Sprints in Complex Projects," in *AGILE 2005 Conference* Denver, CO: IEEE, 2005.

[3] K. Beck, *Extreme Programming Explained: Embrace Change*. Boston: Addison-Wesley, 1999.
[4] H. Takeuchi and I. Nonaka, "The New New Product Development Game," *Harvard Business Review,* 1986.
[5] M. Cohn, *User Stories Applied : For Agile Software Development*: Addison-Wesley, 2004.
[6] J. Sutherland and K. Schwaber, *The Scrum Papers: Nuts, Bolts, and Origins of an Agile Method*. Boston: Scrum, Inc., 2007.
[7] J. Sutherland, A. Viktorov, and J. Blount, "Adaptive Engineering of Large Software Projects with Distributed/Outsourced Teams," in *International Conference on Complex Systems* Boston, MA, USA, 2006.
[8] C. Jones, *Software assessments, benchmarks, and best practices*. Boston, Mass.: Addison Wesley, 2000.
[9] J. Sutherland, A. Viktorov, J. Blount, and N. Puntikov, "Distributed Scrum: Agile Project Management with Outsourced Development Teams," in *HICSS'40, Hawaii International Conference on Software Systems* Big Island, Hawaii: IEEE, 2007.
[10] N. Rathi, "Human resource challenges in Indian software industry: An empirical study of employee turnover," Mercer, 2004.
[11] J. Sutherland, C. Jacobson, and K. Johnson, "Scrum and CMMI Level 5: A Magic Potion for Code Warriors!," in *Agile 2007*, Washington, D.C., 2007.
[12] K. E. Nidiffer and D. Dolan, "Evolving Distributed Project Management," *IEEE Software,* vol. 22, pp. 63-72, Sep/Oct 2005.
[13] M. Poppendieck, "A History of Lean: From Manufacturing to Software Development," in *JAOO Conference*, Aarhus, Denmark, 2005.
[14] W. S. Humphrey, *Introduction to the Personal Software Process*: Addison Wesley, 1996.
[15] M. Cohn, *Agile Estimation and Planning*: Addison-Wesley, 2005.
[16] Danube, "ScrumWorks Pro." vol. 2008 Seattle: Danube, 2008.

Scrum and CMMI Level 5: The Magic Potion for Code Warriors

Jeff Sutherland, Ph.D.
Patientkeeper Inc.
jeff.sutherland@computer.org

Carsten Ruseng Jakobsen
Systematic Software Engineering
crj@systematic.dk

Kent Johnson
AgileDigm Inc.
kent.johnson@agiledigm.com

Abstract
Projects combining agile methods with CMMI[1] are more successful in producing higher quality software that more effectively meets customer needs at a faster pace. Systematic Software Engineering works at CMMI level 5 and uses Lean product development as a driver for optimizing software processes. Valuable experience has been gained by combining Agile practices from Scrum with CMMI.

Early pilot projects at Systematic showed productivity on Scrum teams almost twice that of traditional teams. Other projects demonstrated a story based test driven approach to software development reduced defects found during final test by 40%.

We assert that Scrum and CMMI together bring a more powerful combination of adaptability and predictability to the marketplace than either one alone and suggest how other companies can combine them.

Introduction
One of the trends in the software industry is software projects are more comprehensive and complex while customers at the same time request faster delivery and more flexibility. Successful software development is challenged by the supplier's ability to manage complexity, technology innovation, and requirements change. Customers continually requests solutions faster, better and more cost-effective. Agile and CMMI methods both address these challenges but have very different approach and perspective in methods applied.

Management of complexity requires process discipline, and management of increased speed of change requires adaptability. CMMI primarily provides process discipline and Scrum enhances adaptability. This leads to the question, whether or not it is possible to integrate CMMI and agile practices like Scrum to achieve the benefits from both – or even more?

This paper provides an analysis of the effect of introducing Agile practices like Scrum and story based test driven software development and knowledge gained on what is required to be CMMI compliant, while running an Agile company.

CMMI
The Capability Maturity Model (CMM) has existed since 1991, as a model based on best practices for software development. It describes an evolutionary method for improving an

[1] ® Capability Maturity Model, CMM and CMMI are registered in the U.S. Patent and Trademark Office

organization from one that is ad hoc and immature to one that is disciplined and mature [71]. The CMM is internationally recognized and was developed by the Software Engineering Institute at Carnegie Mellon University, Pittsburgh, USA.

In 2002, a new and significantly extended version called CMMI was announced, where the 'I' stands for 'Integration' [72]. This model integrates software engineering, systems engineering disciplines, and software acquisition practices into one maturity model. CMMI defines 25 process areas to implement. For each process area required goals, expected practices and recommended sub-practices are defined. In addition a set of generic practices must be applied for all processes.

The past 15 years of experience with CMM and CMMI, demonstrates that organizations appraised to higher levels of CMM or CMMI improve the ability to deliver on schedule, cost, and agreed quality. Increasingly, the industry requires suppliers to be appraised to CMM or CMMI level 3 or higher [73]. A number of governmental organizations worldwide, have established CMMI maturity requirements. Recently the Danish Minister of Science proposed regulations to require public organizations to request documentation of their supplier's maturity [74].

Scrum

Scrum for software development teams began at Easel Corporation in 1993 [21] and emerged as a formal method at OOPSLA'95 [22]. A development process was needed to support enterprise teams where visualization of design immediately generated working code. Fundamental problems inherent in software development influenced the introduction of Scrum:

- Uncertainty is inherent and inevitable in software development processes and products - Ziv's Uncertainty Principle [53]
- For a new software system the requirements will not be completely known until after the users have used it - Humphrey's Requirements Uncertainty Principle [58]
- It is not possible to completely specify an interactive system – Wegner's Lemma [54]
- Ambiguous and changing requirements, combined with evolving tools and technologies make implementation strategies unpredictable.

"All-at-Once" models of software development uniquely fit object-oriented implementation of software and help resolve these challenges. They assume the creation of software involves simultaneous work on requirements, analysis, design, coding, and testing, then delivering the entire system all at once [31].

Sutherland and Schwaber, co-creators of Scrum joined forces with creators of other Agile processes in 2001 to write the Agile Manifesto [57]. A common focus on working software, team interactions, customer collaboration, and adapting to change were agreed upon as central principles essential to optimizing software productivity and quality.

CMMI and Agile methods

Soon after publication of the Agile Manifesto in 2001, Mark Paulk principal contributor and editor of Capability Maturity Model Version 1.0 [71], observed that Agile practices are intended

to maximize the benefits of good practice [75, 76]. *"The SW-CMM tells what to do in general terms, but does not say how to do it; agile methodologies provide a set of best practices that contain fairly specific how-to information – an implementation model – for a particular kind of environment."* However, Paulk noted that aligning the implementation of agile methods with the interests of the customer and other stakeholders in a government contracting environment for software acquisition might be an impossible task, where high customer interaction is difficult. Surdu [77] and McMahon [78] reported positive experiences in 2006 using agile processes on government contracts while noting the need for process discipline, good system engineering practices, and development of self-motivated teams. Collaboration with customers was achieved through agile education and negotiation. These studies provide practical confirmation of Paulk's analysis of the applicability of agile practices in a CMM environment.

Paulk [76] points out that *"When rationally implemented in an appropriate environment, agile methodologies address many CMM level 2 and level 3 practices."* Similarly Kane and Ornburn present a mapping of Scrum and XP to CMMI [79] demonstrating that a majority of the CMMI process areas related to Project Management can be addressed with Scrum and the majority of process areas related to software engineering can be addressed with XP. CMMI expects that processes are optimized and perhaps replaced over time and states: *"Optimizing processes that are agile and innovative depends on the participation of an empowered workforce aligned with the business values and objectives of the organization."* [72] (page 49).

We agree with the authors above that Agile methodologies advocate good engineering practices that can be integrated in the CMMI framework, and consider the largest drawback of most Agile methodologies is a limited concept of institution-wide deployment. Institutionalization is key to implementation of all processes in CMMI, and is strongly supported by a set of Generic Practices. It is our belief that these practices could be used to ensure that Agile methodologies are institutionalized in any organization.

Agile methods like Scrum and XP are practical methods that can support different parts of CMMI. Combining Scrum and CMMI practices can produce a more powerful result than either alone and can be done in way where CMMI compliance is maintained. A more detailed analysis of a full implementation of the Scrum development process along with some XP engineering practices used at Systematic shows quantitative results of introducing good agile practices and how to maintain CMMI compliance in an Agile company.

Scrum and CMMI: a magic potion

Systematic was established in 1985 and employs 371 people worldwide with offices in Denmark, USA and the UK. It is an independent software and systems company focusing on complex and critical IT solutions within information and communication systems. Often these systems are mission critical with high demands on reliability, safety, accuracy and usability.
Customers are typically professional IT-departments in public institutions and large companies with longstanding experience in acquiring complex software and systems. Solutions developed by Systematic are used by tens of thousands of people in the defense, healthcare, manufacturing,

and service industries. Systematic was appraised 11 November 2005 using the SCAMPI[SM2] method and found to be CMMI level 5 compliant.
Working at CMMI level 5 brings many advantages. Systematic has first hand experience of reduction in rework by 38% to 42% over earlier levels, estimation precision deviation less than 10%, and 92% of all milestones delivered early or on time. At the same time, extra work on projects has been significantly reduced.
More importantly, Systematic has transformed over twenty years of experience into a unified set of processes used by all software projects. Historical data are systematically collected and analyzed to continuously provide insight into the capability and performance of the organization. The use of a shared common process makes it easier for people to move from project to project and share experiences and lessons learned between projects. Insight into the capability and performance of processes makes it possible to evaluate performance of new processes to performance of existing processes. And this forms the foundation for continuous improvement.

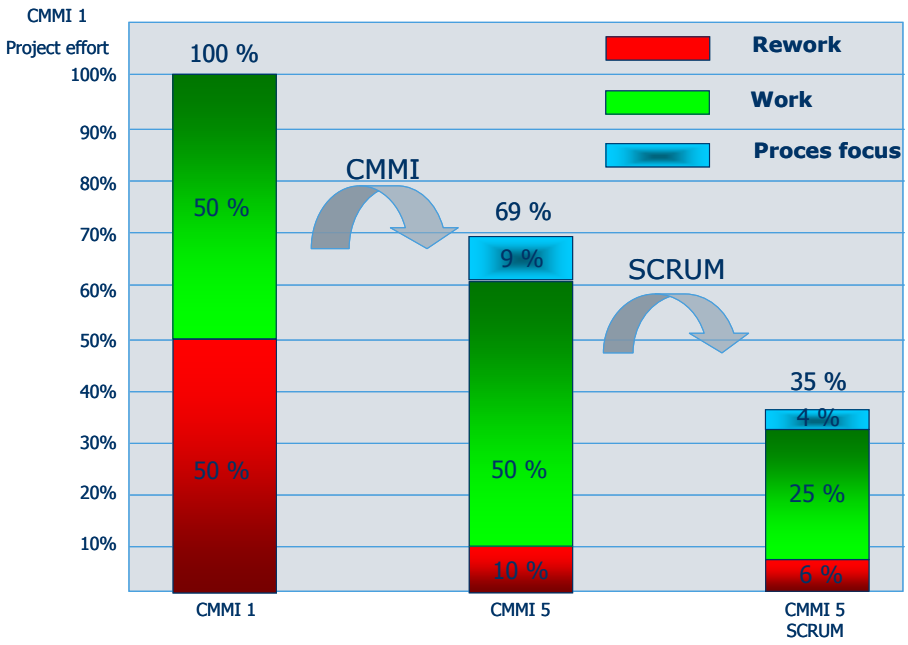


Figure 1: CMMI and Scrum Productivity Gains

In short, Systematic is able to deliver what the customer has ordered on schedule, cost and quality using 69% effort compared to a CMMI Level 1 company [80, 81]. This benefit comes at the minimal cost of 9% process focus in project management and engineering. CMMI Level 5 is increasingly a requirement from customers and key to obtaining large contracts, especially within defence and healthcare. Customers recognize that CMMI Level 5 gives high predictability and better-engineered product for scalability, maintainability, adaptability, and reliability.
Early results indicate that when CMMI traditional processes are optimized using Scrum, the productivity for large projects is doubled and the amount of rework is reduced an additional 40% over that of CMMI Level 5 companies. It is important to note that the optimized process is a mixed process, using traditional CMMI processes to establish a project baseline expressed as a product backlog combined with Scrum as the preferred way to implement the project in iterations

[2] SM Capability Maturity Model Integration, and SCAMPI are service marks of Carnegie Mellon University

of one month Sprints. The combination of the CMMI and Scrum into the optimized CMMI Scrum process includes the proper activities to establish sufficient planning needed by both customer and supplier, and at the same time the flexibility and adaptability provided by Scrum. This combined process is treated similarly to any other process in CMMI.

CMMI provides insight into what processes are needed to maintain a **disciplined** mature organization capable of predicting and improving performance of the organization and projects. Scrum provides guidance for efficient management of projects in a way that allows for high flexibility and **adaptability**. When mixing the two, a magic potion emerges, where the mindset from Scrum ensures that processes are implemented efficiently while embracing change, and CMMI ensures that all relevant processes are considered.

Individually CMMI and Scrum has proven benefits but also pitfalls. An Agile company may implement Scrum correctly but fail due to lack of institutionalization, (see section 0) or inconsistent or insufficient execution of engineering or management processes. CMMI can help Agile companies to institutionalize Agile methods more consistently and understand what processes to address.

A company can comply with CMMI, but fail to reach optimal performance due to inadequate implementation of processes. Scrum and other Agile methodologies can guide such companies towards more efficient implementation of CMMI process requirements.

How Systematic adopted Scrum

Here we describe the generic steps of the process Systematic executed that resulted in the adoption of Scrum, early testing, and story based development.

Identify Business Objectives and Needs. *CMMI states [72] (page 55) that "successful process-improvement initiatives must be driven by the business objectives of the organization". Business objectives and needs are addressed by the strategy of the organization.*

Systematic made a strategic decision to use Lean as the dominant paradigm for future improvements. Lean has demonstrated notable results for many years in domains such as auto manufacturing, and due to it's popularity, has been adapted to other domains, including product and software development. It was expected that adoption of a Lean mindset would facilitate a more efficient implementation of CMMI.

The strategic decision to use Lean as a dominant tool for optimization of processes, is input to CMMI Organizational Process Focus (OPF) and driven by an organizational shared function for process improvements.

Analysis. *Different Lean dialects were evaluated and Lean Software Development [8] was identified as the dialect most relevant to Systematic. Lean Software Development is an agile toolkit. A careful interpretation of the Agile Manifesto shows that this is not necessarily in conflict with CMMI Level 5.*

The Agile Manifesto recognizes that processes, tools, comprehensive documentation, contract negotiation and following a plan have value, but emphasizes people, interactions, working software, customer collaboration and responding to change to have more value. *"The agile methodology movement is not anti-methodology; in fact, many of us want to restore credibility to the word. We also want to restore a balance: We embrace modeling, but not merely to file some diagram in a dusty corporate repository. We embrace documentation, but not to waste reams of paper in never-maintained and rarely used tomes. We plan, but recognize the limits of planning in a turbulent environment." [57]*

Successful application of Lean Software Development - an agile toolkit, depends on the adoption of an agile mindset to supplement the process focus. Systematic values are consistent with the Agile Manifesto, and special focus was placed on the following aspects for new improvements:

Individuals and interactions. Empowerment: the person executing the process is also responsible for updating the process.

Working software over documentation. Critical evaluation of what parts of the documentation or process can be removed or refined to increase the customers perceived value of the activities is essential.

Responding to change. Determining how the process could be improved to support reduced cycle time drove customer value.

Lean competences were established, through handout of books, formal and informal training, and walk-the-talk activities. Project Managers were trained in Lean Software Development, and Mary Poppendieck [8] visited Systematic for a management seminar on Lean Software Development.

This seminar established an understanding of the Agile and Lean mindset. Based on this training, causal dependencies between the principles and tools in Lean Software Development were analyzed, and as a result test practices and reduced cycle time were identified as good candidates for initial activities. They represented a good starting point for implementing Lean and at the same time focused on processes where improvements would have significant effect on efficiency.

Pilot. *Lean advocates that the people performing a process should be responsible and empowered to maintain that process. In the introduction to the CMMI OPF process area CMMI says the same thing.*

An analysis of the causal dependencies in Lean Software Development led to the decision to seek improvements based on the principles of Amplify Learning, Deliver Fast, and Build Integrity In.

Selected projects were asked if they would like to pilot improved processes related to test and reduced cycle time respectively. Project staff were trained in the Lean mindset and asked to

suggest how to adopt Lean into their processes. The result was a selection of Scrum and early testing based on story-based development.

The result of the pilots were two-fold: it confirmed the general idea of using Lean mindset as source for identification of new improvements, and secondly it provided two specific successful improvements showing how agile methods can be adopted while maintaining CMMI compliance.

Implementation. *It was decided to adopt Scrum and story based software development in the organization. Process Action Teams (PATs) were formed to integrate the experience and knowledge gained from the pilots, into the processes shared by all projects in the organization. The PATs were staffed with people that would be expected to execute the new process when released.*

The largest change to project planning is that features and work are planned in sufficient detail as opposed to a complete initial detailed analysis. The result is a Scrum Product Backlog with a complete prioritized list of features/work for the project. All features have a qualified estimate, established with a documented process and through the use of historical data, but the granularity of the features increase as the priority falls. The uncertainty that remains is handled through risk management activities.

The primary change to project execution processes, was to integrate Scrum as method for completing small iterations (Sprints), on a selected subset of the work with highest priority. This work verified that Scrum could be adopted in project management while maintaining CMMI compliance. This is important because, one of the first steps to embrace change is to ensure that project management processes support and allow agility. In addition the people executing the process were trained and certified as ScrumMasters by Jeff Sutherland [82], who also did a management seminar on Scrum at Systematic. Concurrent to the above pilots, Lean was considered by all projects and shared functions as one of several ways to identify possible improvements.

Result. *The first step for Systematic in adapting a Lean mindset resulted in the adoption of Scrum and story based development as the recommended way of working. Systematic provides a default implementation of a Projects Defined Process (PDP) known as PDP Common. The PDP Common has been updated to integrate Scrum and story based development into the relevant processes.*

The apparent result of adopting agile methods into existing CMMI compliant processes, has led to integration of processes or process areas that initially were implemented separately. The new processes are more efficient, and the changes have improved quality, customer and employee satisfaction.

Risk. *Some of the risks of applying Agile mindset to a CMMI organization include*:

- Degrading CMMI compliant processes to non-compliance.

- Local optimizations increasing project efficiency at the expense of inefficiency at the organizational level, e.g. due to lack of organizational coordination.

These risks were handled by a central process team that kept the organization on track with respect to the risks and change management. The process team was responsible for:

- Build and share competencies on Lean, Agile and Scrum with the organization.
- Define and communicate vision, constraints and measures for adoption of a Lean mindset.
- Encourage and empower different parts of the organization to challenge current process implementations with a Lean mindset, in search of improvement opportunities.
- Collect experiences from the organization and consolidate improvements at the organizational level.

The dominant risk for failure in adapting Lean is insufficient understanding or adoption of Lean or Agile mindset. Systematic has addressed this risk by inviting Jeff Sutherland and Mary Poppendieck to Systematic to establish a good understanding of Lean, Scrum and Agile.

Systematic experience from pilots

In a period of approximately 4 months, two small projects piloted Scrum and early testing in story based development.

Scrum. *The first pilot was initiated on a request for proposal, where Systematic inspired by Lean principles suggested a delivery plan with bi-weekly deliveries and stated explicit expectations to customer involvement and feedback.*

One of the main reasons that Systematic was awarded the contract was the commitment to deliver working code bi-weekly and thereby providing a very transparent process to the customer. During project execution, a high communication bandwidth was kept between the team, the customer and users. This was identified as one of the main reasons for achieving high customer satisfaction.

The delivery plan and customer involvement resulted in early detection of technological issues. Had a traditional approach been used these issues would have been identified much later with negative impacts on cost and schedule performance.

However, productivity of this small project was at the expected level compared to the productivity performance baseline for small projects. Another small project using Scrum shows a similar productivity and the same indications on high quality and customer satisfaction.

At Systematic, productivity for a project is defined as the total number of lines of code produced divided by the total project effort spent in hours. These numbers are gathered from the configuration and version control system. Data are attributed with information related to programming language, type of code: new, reuse or test. This definition of productivity has been chosen because it provides sufficient insight and is very simple and efficient to collect. Systematic has established and maintains a productivity performance baseline (PPB) for productivity compared to project size estimated in hours, from data collected on completed projects [83]. The data shows that productivity is high on small projects and declines with the

size of the project with traditional CMMI Level 5. The productivity performance baseline in Systematic is divided into two groups: small projects less than 4000 hours and large projects above 4000 hours. Productivity of small projects is 181% the productivity of large projects. When comparing the projects using Scrum to the current productivity baseline it is seen that productivity for small projects is insignificantly changed, but the productivity for large projects shows 201% increase in productivity. As mentioned above, the large projects did additional improvements, and it is therefore not possible to attribute the benefit solely to Scrum. However the people involved all agree that Scrum was a significant part of this improvement.

There is a strong indication that large projects in Systematic using Scrum will double productivity going forward. Small projects in Systematic already show a high productivity. We believe that this is because small projects in Systematic always have been managed in a way similar to Scrum. However quality and customer satisfaction seems to be improved and we believe this is because Scrum has facilitated a better understanding of how small projects are managed efficiently.

Early testing. *Two different approaches were used to facilitate early test. One large project decided to use a story based approach to software development and another project decided to focus on comprehensive testing during development.*

The idea of story-based development was to subdivide features of work, typically estimated to hundreds of hours of work into smaller stories of 20-40 hours of work. The implementation of a story followed a new procedure, where the first activity would be to decide how the story could be tested before any code was written. This test could then be used as the exit criteria for implementation of the story.

In order to ensure that the new procedure was followed, the procedure included a few checkpoints where an inspector would inspect the work produced, and decide whether or not the developer could proceed to the next activity in the procedure. These inspections are lightweight, and could typically be done in less than 5 minutes.

Many benefits from story-based development were immediately apparent. The combination of a good definition of when a story was complete, and early incremental testing of the features, provided a very precise overview of status and progress for both team and other stakeholders. Developing a series of small stories rather than parts of a big feature is more satisfactory, and creates a better focus on completing a feature until it fulfills all the criteria for being “done”. This project finished early, and reduced the number of coding defects in final test by 38% compared to previous processes.

The project using comprehensive testing placed test specialists together with the developers. As in the story based approach, this caused discussion and reflection between testers, developers, user experience engineers and software architects, before or very early in the development of new functionality. As a consequence the amount of remaining coding defects in final test were reduced by 42%.

Based on these two projects test activities should be an integrated activity through out the projects lifetime. Scrum inherently supports this, through cross-functional teams and frequent deliveries to the customer.

Real needs. *A customer sent a request for proposal on a fixed set of requirements. When Systematic responded, we expressed our concern that the scope and contents expressed in the requirements were beyond the customer's real needs.*

Systematic decided to openly share the internal estimation of the requirements with the customer, for the purpose of narrowing scope by removing requirements not needed or too expensive compared to the customers budget. The customer agreed to re-evaluate the requirement specification, and the result was that requirements and price were reduced by 50%.

This experience supports results in a Standish Group Study reported at XP2002 by chairman Jim Johnson, showing that 64% of features in a fixed price contract are never or rarely used by end-users.

We believe that this illustrates how important it is to have a high communication bandwidth with the customer, in order to find out what the real needs are. Success is not achieved by doing the largest project, but by doing the project that provides the most value for the customer, leaving time for software developers to work with other customers with real needs. It gives motivation to developers to provide solutions to real need, which in turn benefits dedication and productivity. Even though this experience is related to activities before the project is started, the challenge of maintaining close communication with the customer, to ensure that the project delivers the most value within the agreed constraints, continues and is strongly supported by Scrum.

Guide for mixing CMMI and Agile

The previous section has described how Systematic, an organization appraised to CMMI Level 5, has adopted agile methods. This section presents our advice to the agile organizations on how to adopt the concept of institutionalization.

How CMMI can improve Agile

Our focus is on using CMMI to help an organization institutionalize Agile Methods. We have all heard Agile Methods described by some as just another disguise for undisciplined hacking and of some individuals who claim to be Agile just because they "don't document." We believe the value from Agile Methods can only be obtained through disciplined use. CMMI has a concept of *Institutionalization* that can help establish this needed discipline.

Institutionalization is defined in CMMI as "the ingrained way of doing business that an organization follows routinely as part of its corporate culture." Others have described institutionalization as simply "this is the way we do things around here." Note that institutionalization is an organizational-level concept that supports multiple projects.

CMMI supports institutionalization through the Generic Practices (GP) associated with all process areas. For the purposes of our discussion, we will look at the 12 generic practices

associated with maturity levels 2 and 3 in the CMMI [72] pp. 39-44 and how they might help an organization use Agile Methods. We have paraphrased the generic practices (**shown in bold text below**) to match our recommended usage with Agile Methods. In CMMI terms, the projects in an organization would be *expected* to perform an activity that accomplished each of these generic practices. We have used Scrum as the example Agile Method to describe some of the activities that relate to these practices.

Establish and maintain an organizational policy for planning and performing Agile Methods (GP 2.1). The first step toward institutionalization of Agile Methods is to establish how and when they will be used in the organization. An organization might determine that Agile Methods will be used on all projects or some subset of projects based on size, type of product, technology, or other factors. This policy is a way to clearly communicate the organization's intent regarding Agile Methods. In keeping with the Agile Principle of face-to-face conversions at "all hands meeting" or a visit by a senior manager during a project's kick off could be used to communicate the policy.

Establish and maintain the plan for performing Agile Methods (GP2.2). This practice can help ensure that Agile Methods do not degrade into undisciplined hacking. The expectation is that Agile Methods are planned and that a defined process exists and is followed. The defined process should include a sequence of steps capturing the minimum essential information needed to describe what a project really does. The plan would also capture the essential aspects of how the other 10 generic practices are to be implemented in the project. In Scrum, some of this planning is likely to be captured in a product backlog and/or sprint backlog, most likely within a tool as opposed to a document.

Provide adequate resources for performing Agile Methods (GP2.3). Every project wants, needs, and expects competent professionals, adequate funding, and appropriate facilities and tools. Implementing an activity to explicitly manage these wants and needs has proved useful. In Scrum, for example, these needs may be reviewed and addressed at the Sprint Planning Meeting and reconsidered when significant changes occur.

Assign responsibility and authority for performing Agile Methods (GP 2.4). For a project to be successful, clear responsibility and authority need to be defined. Usually this includes a combination of role descriptions and assignments. The definitions of these roles identify a level of responsibility and authority. For example, a Scrum Project would assign an individual or individuals to the roles of Product Owner, ScrumMaster, and Team. Furthermore, the roles in the Team are likely to include a mix of domain experts, system engineers, software engineers, architects, programmers, analysts, QA experts, testers, UI designers, etc. Expertise in the Team is likely to include a mix of domain experts, system engineers, software engineers, architects, programmers, analysts, QA experts, testers, UI designers, etc. Scrum assigns the team as a whole the responsibility for delivering working software. The Product Owner is responsible for specifying and prioritizing the work. The ScrumMaster is responsible for assuring the Scrum process is followed. Management is responsible for providing the right expertise to the team.

Train the people performing Agile Methods (GP 2.5). The right training can increase the performance of competent professionals and supports introducing new methods into an

organization. People need to receive consistent training in the Agile Method being used in order to ensure institutionalization. This practice includes determining the individuals to train, defining the exact training to provide, and performing the needed training. Training can be provided using many different approaches, including programmed instruction, formalized on-the-job training, mentoring, and formal and classroom training. It is important that a mechanism be defined to ensure that training has occurred and is beneficial.

Place designated work products under appropriate level of configuration management (GP 2.6). The purpose of a project is to produce a deliverable product (or products). This product is often a collection of a number of intermediate or supporting work products (code, manuals, software systems, build files, etc.). Each of these work products has a value and often goes through a series of steps that increase their value. The concept of configuration management is intended to protect these valuable work products by defining the level of control, for example, version control or baseline control and perhaps multiple levels of baseline control to use within the project.

Identify and involve the relevant stakeholders as planned (GP 2.7). Involving the customer as a relevant stakeholder is a strength of Agile Methods. This practice further identifies the need to ensure that the expected level of stakeholder involvement occurs. For example, if the project depends on customer feedback with each increment, build, or sprint, and involvement falls short of expectations it is then necessary to communicate to the appropriate level, individual, or group in the organization to allow for corrective action. This is because corrective action may be beyond the scope of the project team. In advanced Scrum implementations, this is often formalized as a MetaScrum [40] where stakeholders serve as a board of directors for the Product Owner.

Monitor and control Agile Methods against the plan and take appropriate corrective action (GP 2.8). This practice involves measuring actual performance against the project's plan and taking corrective action. The direct day-to-day monitoring is a strong feature of the Daily Scrum Meeting. Further, examples of this can be seen in Scrum with the use of the Product Burndown Chart showing how much work is left to do at the beginning of each Sprint and the Sprint Burndown Chart showing the total task hours remaining per day. Scrum enhances the effectiveness of the plan by allowing the Product Owner to inspect and adapt to maximize ROI, rather than merely assuring plan accuracy.

Objectively evaluate adherence to the Agile Methods and address noncompliance (GP2.9). This practice is based on having someone not directly responsible for managing or performing project activities evaluate the actual activities of the project. Some organizations implement this practice as both an assurance activity and coaching activity. The coaching concept matches many Agile Methods. The ScrumMaster has primary responsibility for adherence to Scrum practices, tracking progress, removing impediments, resolving personnel problems, and is usually not engaged in implementation of project tasks. The Product Owner has primary responsibility for assuring software meets requirements and is high quality.

Review the activities, status, and results of the Agile Methods with higher-level management and resolve issues (GP2.10). The purpose of this practice is to ensure that higher-

level management has appropriate visibility into the project activities. Different managers have different needs for information. Agile Methods have a high level of interaction, for example, Scrum has a Sprint Planning Meeting, Daily Scrum Meetings, a Sprint Review Meeting, and a Sprint Retrospective Meeting. Management needs are supported by transparency of status data produced by the Scrum Burndown Chart. This, in combination with defect data can be used to produce a customized management dashboard for project status. Management responsibilities are to (1) provide strategic vision, business strategy, and resources, (2) remove impediments surfaced by Scrum teams that the teams cannot remove themselves, (3) ensure growth and career path of staff, and (4) challenge the Scrum teams to move beyond mediocrity. The list of impediments generated by the Scrum teams is transparent to management and it is their responsibility to assure they are removed in order to improve organizational performance.

Establish and maintain the description of Agile Methods (GP 3.1). This practice is a refinement of GP2.2 above. The only real difference is that description of Agile Methods in this practice is expected to be organization-wide and not unique to a project. The result is that variability in how Agile Methods are performed would be reduced across the organization; and therefore more exchange between projects of people, tools, information and products can be supported.

Collect the results from using Agile Methods to support future use and improvement the organization's approach to Agile Methods (GP 3.2). This practice supports the goal of learning across projects by collecting the results from individual projects. The Scrum Sprint Retrospective Meeting could be used as the mechanism for this practice.

All of these generic practices have been useful in organizations implementing other processes. We have seen that a number of these generic practices have at least partial support in Scrum or other Agile Methods. We believe that implementing these practices can help establish needed discipline to any Agile Method.

Critiques of CMM

In research funded by the Danish government, Rose et. al. surveyed the literature on critiques of CMM [84]. They observed that the chief criticism of CMM is not the process itself, but the effects of focus on process orientation. While side effects of process focus may be viewed as simply poor CMM implementation, organizations with heavyweight processes are highly prone to poor execution.

As with any other model, good and bad implementations of CMM exist. We believe that bad implementations are one of the main reasons for the existence of many negative criticisms of CMM. Such implementations are often characterized as in the table below, whereas many good CMM implementations address most of the criticism.

One way to enhance chances for a good CMM or CMMI implementation is to use Scrum. Applying Scrum and agile mindset while implementing CMMI will help to recognize that CMMI addresses people and technology aspects, in a way fully supportive of an agile mindset. More importantly, the work in this paper has shown that the mix of CMMI and Scrum blends a magic potion for software development that is even better than the sum of the two alone.

We acknowledge that the CMM criticism listed in the table below exist, but from our knowledge of CMMI we consider it to be incorrect. But a bad implementation of CMMI may be perceived this way. Even though good CMMI implementations can be done without agile methods, the table shows that Scrum will contribute with a beneficial focus on issues stemming from "bad" CMMI implementation.

CMM criticism	**Scrum support**
CMM reveres process but ignores people.	Scrum is the first development process to treat people issues the same as other project management issues [85].
Does not focus on underlying organizational problems that should be solved.	A primary responsibility of the ScrumMaster is to maintain and resolve an impediment list that contains organizational issues, personal issues, and technical problems.
Ignores quality in the software product assuming an unproven link between quality in the process and quality in the resulting product. Differing project and organizational circumstances may mean that a process that delivers a good product in one context delivers a poor product in another context.	The Scrum Product Owner is responsible for continuously reprioritizing the Product Backlog to maximize business value in current context.
Lack of business orientation	The primary focus of Scrum is on delivering business value.
Poor awareness of organizational context.	Creation and prioritization of features, tasks, and impediments is always done in organizational context by inspected and adapting.
Ignores technical and organizational infrastructures.	Daily inspection and adaptation in Scrum meetings focuses on technical and organizational issues.
Encourages an internal efficiency focus and thus market and competition blindness.	Focus is on delivering business value. Type C Scrum allows an entire company to dominate a market segment through inspecting and adapting in real time to competition [40].

Conclusions

This paper shows that CMMI and Scrum can be successfully mixed. The mix results in significantly improved performance while maintaining compliance to CMMI Level 5 as compared to performance with either CMMI or Scrum alone.

Scrum pilot projects showed significant gains in productivity and quality over traditional methods. These results led to an ROI based decision to more widely introduce Scrum and consider other Agile practices in Systematic. Scrum now reduces every category of work (defects, rework, total work required, and process overhead) by almost 50%.

This paper has outlined how Systematic adopted Scrum and story based development into its CMMI processes inspired from a strategic focus on Lean. For Agile companies the article has presented how Generic Practices from CMMI can be used to institutionalize agile practices. Furthermore the article has presented Lean Software Development [8] as an operational tool to identify improvement opportunities in a CMMI 5 company.

We think companies in defense, aerospace, and other industries that require high maturity of processes, should carefully consider introducing Agile practices into the workplace and all software companies should consider introducing CMMI practices into their environment. Our recommendation to the Agile community is to use the CMMI generic practices from CMMI Level 3 to amplify the benefits from Agile methods. The efficiencies of agile practice can lower the cost of CMMI process improvements making the benefits more widely available. Our recommendation to the CMMI community is that Agile methods can fit into your CMMI framework and will provide exciting improvements to your organization.

Mature Agile with a twist of CMMI

Carsten Ruseng Jakobsen
Systematic Software Engineering
crj@systematic.dk

Kent Aaron Johnson
AgileDigm, Incorporated
kent.johnson@agiledigm.com

Abstract

Systematic is an agile company working at CMMI level 5, where the default way of working is based on Scrum and story based early testing development. Solid experiences in combining CMMI with Scrum and story based development, has shown that the mix provides strong synergies [2] and insights into what CMMI practices fit and amplify the execution of Scrum and story based early testing development

This paper presents specifically how agile methods like Scrum are successfully combined with CMMI. CMMI provides solid support for what disciplines to consider. When applied the disciplines create a focus on important aspects of agile methods that perhaps are not normally elaborated, for example how to ensure a proper quality of a product backlog or how to ensure a proper "production line" for the project. This guidance may not be needed for small agile projects, but as the agile movement continues to grow, and is used for larger and more complex projects, agile projects will need to address these issues related to increased size and complexity.

The experiences from combining CMMI and Scrum have led Systematic to identify examples of explicit guidance from CMMI that help to execute normal Scrum activities even better.

These activities can be implemented in the spirit of the agile manifesto and principles and by doing so agile methods can be augmented and matured to ensure that even larger and more complex projects in the future can and will benefit from agile - with a twist of CMMI.

Introduction

This paper presents the experiences on how CMMI amplifies Agile and recommends a subset of activities an agile project could adopt from CMMI to improve performance. Agile purists and small agile projects may find these activities non-agile or counter-productive; however in larger and/or distributed projects these activities will prove to be invaluable.

The paper does not describe how to mature an organization from CMMI level 1 to CMMI level 2, but the activities described could be part of such a change. Rather this paper highlights a set of activities that could be considered the glue between agile only projects based on Scrum and the disciplines expected from projects and organizations working toward CMMI level 2 or 3.

This paper presents how "a twist of CMMI" can help establish a more solid framework for agile projects to support even more complex projects by adopting some of the practices from the CMMI.

Context for the experiences

Systematic was established in 1985 and employs more than 450 people worldwide with offices in Denmark, Finland, USA and the UK. It is an independent software and systems company focusing on complex and critical IT solutions within information and communication systems. Often these systems are mission critical with high demands on reliability, safety, accuracy and usability.

Customers are typically professional IT-departments in public institutions and large companies with longstanding experience in acquiring complex software and systems. Solutions developed by Systematic are used by tens of thousands of people in the defense, healthcare, manufacturing, and service industries. Systematic was appraised 11 November 2005 using the SCAMPI[SM] method and found to be CMMI level 5 compliant. During 2006 Systematic adopted Scrum and a story based early testing approach to software development.

[SM] Capability Maturity Model Integration, and SCAMPI are service marks of Carnegie Mellon University

CMMI provides insight into what processes are needed to maintain a **disciplined** mature organization capable of predicting and improving performance of the organization and projects. Scrum provides guidance for efficient management of projects in a way that allows for high **flexibility** and **adaptability**. When mixing the two, a magic potion emerges, where the mindset from Scrum ensures that processes are implemented efficiently while embracing change, and CMMI ensures that all relevant processes are considered with proper discipline.

Individually CMMI and Scrum has proven benefits, but also pitfalls. An agile company may implement Scrum correctly, but fail to obtain real benefits due to lack of consistent and sufficient execution of engineering or management processes. CMMI can help agile companies to institutionalize agile methods more consistently and understand what processes to address.

A company can comply with CMMI, but fail to reach optimal performance due to inadequate implementation of processes. Scrum and other agile methods can guide such companies towards more efficient implementation of CMMI process requirements.

Systematic has gained valuable experiences in combining Scrum and CMMI that are relevant both for projects in a CMMI and agile context.

Experiences from mixing

The first major experience from working with Scrum in a CMMI context is that CMMI embraces Scrum. CMMI has more practices and support for initial project planning and for final delivery and project closure. In Scrum terms, CMMI suggests activities before and after sprints are executed on the product backlog.

From a CMMI perspective, the initial Scrum product backlog is created during project planning and during project execution; sprints are executed and the product backlog is updated. This logically splits planning into two parts: overall CMMI project planning and detailed agile planning through Scrum.

This separation has led to overall planning where work is planned in sufficient detail as opposed to a complete decomposition. The overall planning produces a set of overall project plans and a Scrum product backlog where a complete list of prioritized features or work for the project is managed.

The primary change to project execution processes, was to integrate Scrum as the method for completing small iterations (sprints), on a selected subset of the work with highest priority.

Systematic experience indicates that this mix of CMMI, Scrum, and agile is beneficial, because

- CMMI planning can be considered a kind of disciplined sprint zero, where it is ensured that an optimal framework for the project is established, including a high quality product backlog, a production line definition, and well known targets and vision for the project as a whole.
- CMMI risk management proactively addresses possible impediments before they are encountered by the team.
- CMMI quality planning specifies more accurately and efficiently the quality targets of the project and helps developers to a better interpretation of completion criteria and sprint goals.
- CMMI will ensure that the project is tracked as a whole allowing the Scrum Team to concentrate on current sprint, knowing that they periodically are informed of overall project status.
- Scrum requires discipline regarding automatic test, a nightly build, and integration. CMMI supports this need for discipline and has led some projects at Systematic to monitor "fix-time after failed builds" for more than a year. The measure has proven to be cheap to establish, easy to understand, and therefore facilitating good habits.
- CMMI expects the project to seek objective measures of performance of the project's processes. In Scrum progress (of sprints) is primarily measured through the sprint burn down chart and the sprint review meeting. The project manager tracks the project as a whole based on selected measures within key areas like, product size, earned value, schedule, and quality. CMMI project planning provides good overall plans for the complete project where each completed sprint is very valuable input.
- CMMI ensures that agile methods are institutionalized, including
    - Consistent implementation throughout the organization and continuous improvement, e.g. Systematic Scrum Guidelines, story inspection checklist.
    - Role based training of all roles, e.g. Scrum Master and Product Owner.

When Systematic adopted Scrum, the roles Scrum Master, Product Owner, and Team were introduced. Most projects in Systematic have a person who communicates and understands the customer. In Systematic this person is

appointed the role Product Owner. In our experience the Product Owner is often also the Project Manager, but could also be Software Architect, or User Experience Engineer. The Scrum Masters are in most cases equivalent with the team leader roles in Systematic. Projects in Systematic are staffed with people working full time on the project and the team is co-located.

How to establish a better initial product backlog

Experience. Project Planning in CMMI is a disciplined and comprehensive Sprint Zero.

In order to run a good Scrum, it is vital to have good product backlog. When Systematic adopted Scrum, the project planning process was updated to produce an initial product backlog. Expected CMMI practices include decomposition of work into manageable pieces that are estimated and analyzed for dependencies, planning of stakeholder involvement, and total project risk assessment.

In addition to the product backlog a set of overall project plans are established. Agile teams talk about a sprint zero to establish the foundation for the team to do efficient sprinting. Project Planning in CMMI can be perceived as a sprint zero to produce a coherent set of plans, that will help improve execution of the product backlog. Such plans cover topics like, stakeholder management, milestone and delivery schedules, cost estimates, and quality.

The initial version of these plans are typically established within few weeks after project initiation and will focus on the most certain elements of the projects plan, leaving more uncertain parts to be managed on the product backlog.

Figure 1 shows the main activities performed to establish the project plan. The sequence shown in the figure is advisory. The same activity may be performed multiple times, and often many activities are performed simultaneously. Each of the activities are supported by short step-wise descriptions of how the activity normally is performed, what inputs are required and what output are produced.

The descriptions are based on lessons learned and best practices from all projects within Systematic, and are organized with short outlined descriptions at the top, and detailed guidelines and templates at the bottom.

This provides the team with solid support for establishing the projects plans. The amount of time spent to establish project plans varies from project to project, but for most projects, initial project planning can be completed within weeks. One of the reasons for this is the handling of uncertain scope through Scrum combined with proper risk management from CMMI.

Concurrent to the initial planning, the overall solution architecture and product quality objectives are elaborated and documented by software architects and lead developers.

Before the project moves from planning (sprint zero) to project execution (sprinting) the project manager validates that the overall project objectives and plans are achievable and realistic, and that planning has reduced project risks sufficiently.

Many of the activities in figure 1 are in line with the intensions of Scrum, the main difference is that in CMMI these activities are elaborated and documented.

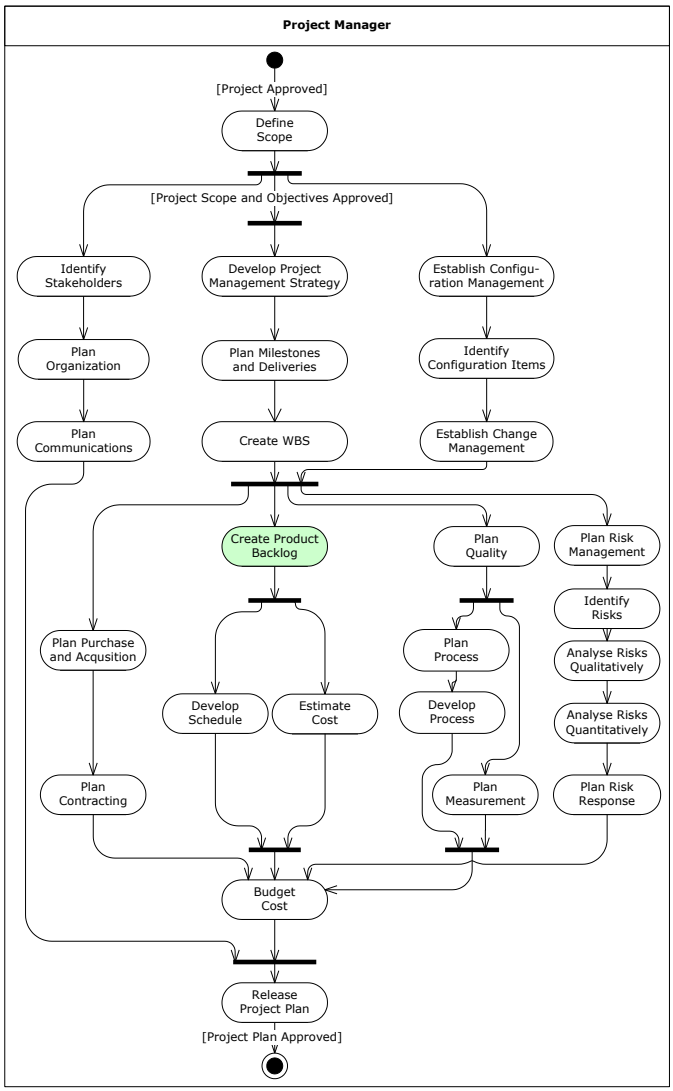


Figure 13 Overview of CMMI planning activities

Experience. Two levels of planning and tracking: Project as a whole and each sprint.

Sprinting on the product backlog is started, when the project plans and solution architecture are approved by senior management.

From this point tracking of the project is done at two different levels concurrently. The project manager tracks the projects overall plans and team(s) tracks progress of active sprint(s).

The plans established with planning activities provides a better context for defining sprint visions and goals, compared to projects only using a Sprint backlog. They also allow the team to focus on the sprint, because they can rely on the project manager tracking the overall progress.

Risks and impediments

Experience: Planning and risk management activities reduces risk of product backlog

The first draft product backlog is assessed for risk by the team. Asking the team to estimate the products backlog using 3-point estimates for effort will reveal the most uncertain parts of the work in the Product Backlog. Conducting Risk Identification meetings will identify other important risks, and allow proactive mitigation to be initiated.

Experience: Risk management can proactively prevent impediments

Scrum has a strong focus on removing impediments as soon as they are identified, however CMMI risk management activities focus on proactively identify some of these impediments as risks, and through mitigation eliminate them before they occur as an impediment in the future.

The distinction between risk and impediment is that risk describe a problem that may occur, whereas an impediment is problem that has occurred and is impacting planned progress.

Risk management activities are easily integrated with Scrum activities. During project planning the project plans and solution architecture are inputs for initial identification of risks. During project execution, bi-weekly meetings of 10-15 minutes are arranged, where the status of known risks is reported and new risks are identified.

It is our experience that these risk management meetings should be kept outside the daily scrum meeting. New risks may be reported on the daily scrum, in which case the risk manager will just take a note.

How to ensure high quality

Scrum is designed to produce high quality in terms of perceived and conceptual integrity where short iterations and sprint review customers are main drivers for high quality.

Experience: *Explicit quality plans improves helps the team to build the right quality in*

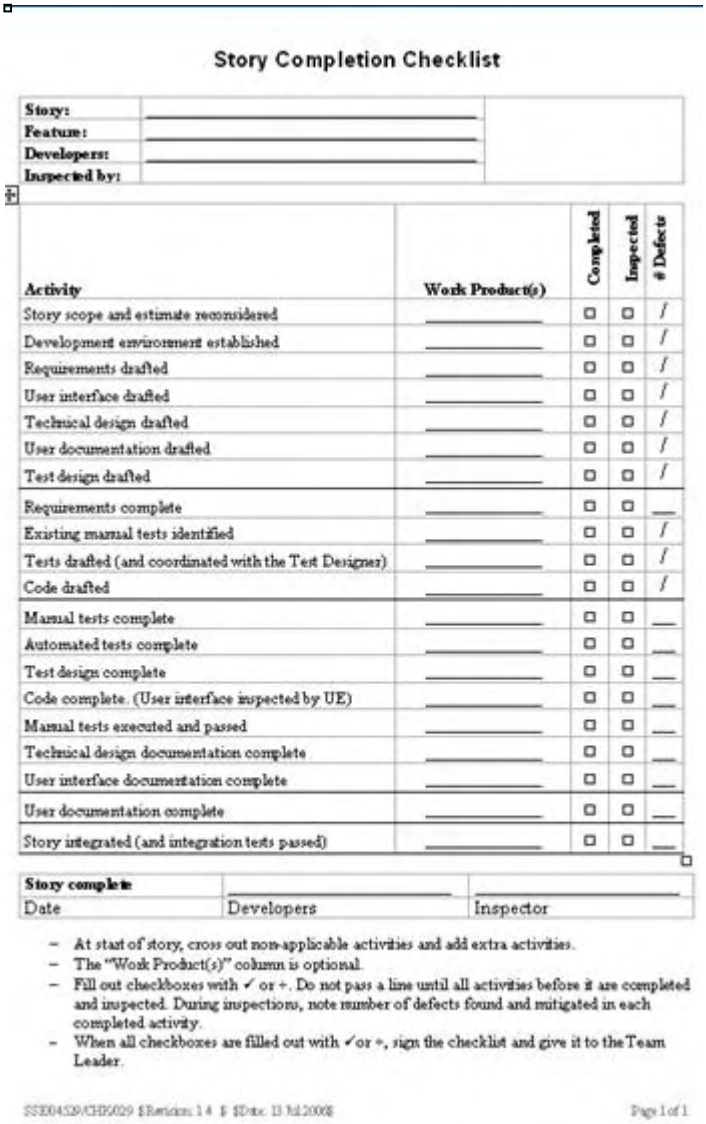
One of the results of planning is a quality assurance schedule (QAS), where it is outlined what quality activities will be used to ensure the quality objectives are achieved. The QAS may specify

- What stories are subject to inspection
- What code is subject to review
- What documents are subject to what types of review
- What unit test and automatic test is produced
- What is included in the acceptance test

A typical QAS document is only a few pages long, but the above descriptions can help a scrum team to elaborate and understand the definition of done.

Experiece: Use checklist to ensure quality of stories

One of the important aspects of Systematic story based development method was to ensure focus on early test.



Story Completion Checklist

Story:	
Feature:	
Developers:	
Inspected by:	

Activity	Work Product(s)	Completed	Inspected	# Defects
Story scope and estimate reconsidered		☐	☐	/
Development environment established		☐	☐	/
Requirements drafted		☐	☐	/
User interface drafted		☐	☐	/
Technical design drafted		☐	☐	/
User documentation drafted		☐	☐	/
Test design drafted		☐	☐	/
Requirements complete		☐	☐	___
Existing manual tests identified		☐	☐	/
Tests drafted (and coordinated with the Test Designer)		☐	☐	/
Code drafted		☐	☐	/
Manual tests complete		☐	☐	___
Automated tests complete		☐	☐	___
Test design complete		☐	☐	___
Code complete. (User interface inspected by UE)		☐	☐	___
Manual tests executed and passed		☐	☐	___
Technical design documentation complete		☐	☐	___
User interface documentation complete		☐	☐	___
User documentation complete		☐	☐	___
Story integrated (and integration tests passed)		☐	☐	___

Story complete		
Date	Developers	Inspector

- At start of story, cross out non-applicable activities and add extra activities.
- The "Work Product(s)" column is optional.
- Fill out checkboxes with ✓ or +. Do not pass a line until all activities before it are completed and inspected. During inspections, note number of defects found and mitigated in each completed activity.
- When all checkboxes are filled out with ✓or +, sign the checklist and give it to the Team Leader.

SSE04529/CHK029 $Revision: 1.4 $ $Date: 13 Jul 2008

Page 1 of 1

Figure 14 Story inspection checklist

The quality of stories are generally ensured by focus on early specification of test and by getting somebody else to look at the work done.

Developing a story includes many different activities, that need to be structured to some degree.

The Story Completion Checklist accomplishes these goals, by structuring activities and defining when work must be inspected by an inspector.

The activities in the checklist all have a short 5-10 line description in a procedure called "Execute story", that clarifies why and how the activity is performed.

The inspector role is often appointed to a lead developer, and this way the inspection also serves as an opportunity for knowledge sharing between experienced and less experienced developers.

The Story Completion Checklist ensures quality at the story level and makes it easier for the developer at the same time.

Test, integration, release and configuration management

Scrum promotes short iterations, e.g. one sprint per month, and this in turn drives the need for efficient configuration management, test, integration and release.

CMMI helps with:

- Establish standards for production line, including standard setups for build- and test servers
- Establish discipline on criteria for integration
- Measures to objectively evaluate performance
- Disciplines to maintain integrity of configuration management system, builds, and releases.

Experience: Automated test is a must in order to do one month sprints

When the sprint duration is one month, all tests must be automated to the extent possible. It is an integrated part of developing a story, to also implement the automated test verifying the story. Automated test are used on the teams shared repository and run every time a developer commits code to the shared build server.

Experience: Automated test and integration must be supported by a standard production line

Every project needs this infrastructure and therefore we have established standard production line setups, allowing projects to get started faster.

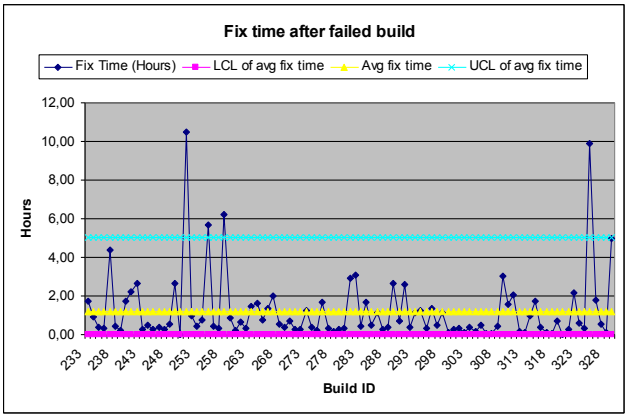
Experience: Continuous integration must be supported with discipline for check in

In order to avoid chaos when developers continuously integrate with each other, we have defined the following criteria for check in of code to the integration repository: The test must run smoothly in the developers sandbox and the code must comply with the code standard checked with FxCop (a static code analysis tool).

Experience: Focus on "fix-time after failed build" drives good discipline in project

Using standard infrastructure setups allows for efficient data collection and analysis. In particular Systematic has been inspired from Lean thoughts on flow and jidoka (stopping the production line when an error is detected).

We want our projects to be able to deliver on a daily basis, and hence that unresolved failed builds are fixed within a working day. We use CruiseControl (a build management tool) to signal all developers when a build fails. We also monitor the objective by analyzing data from the build servers using control charts like the one shown below.



Many projects have achieved this one work day objective, merely by the focus on the measure. The objective is easy to understand, and presenting the information in CruiseControl and control charts has established a good habit of fixing broken builds immediately when they fail.

Experience: Periodic audit of Configuration Management system builds good habits

As part of every sprint delivery a work product evaluation (WPE) is conducted and for every delivery to the customer a functional configuration audit (FCA) is conducted. The purpose is to ensure that the build product is correct and complete.

WPE and FCA are executed and documented by filling out check-lists that helps to ensure that configuration management activities has been executed correctly for the build or release. Usually these activities can be accomplished within one or two hours.

The experience is that this checking of the configuration management system on a monthly basis, builds good habits on the team to remember configuration management disciplines, and as a result builds and releases are complete and correct, and may be re-created in the future should the need arise.

Agile with a twist of CMMI

In [2], [3] we described how the generic practices from CMMI can be used to institutionalize Scrum in your organization. In the following we present our recommendations to activities an agile project could consider adopting. These activities are inspired by the mandatory goals and expected practices from a subset of CMMI process areas.

We recommend the following activities to agile projects:

1. Establish your own sprint zero, and include activities in item 2-6 below in it.
2. Use Risk Management to proactively address risks before they are identified as impediments
3. Decompose requirements into features on the product backlog. Prepare the product backlog by decomposing the highest prioritized features to stories allowing for efficient sprint planning. (This defines what you are really going to do.)
4. Use 3-point effort estimates on elements of the product backlog during initial planning.
5. Analyze dependencies, stakeholders, risk on elements of product backlog.
6. Establish milestone and delivery plan and their initial relationship to product backlog.
7. Use Story Completion Checklist to maintain high quality of stories produced.
8. Decide and communicate quality objectives including, what code and documentation to formally review to elaborate definition of done.
9. Establish standards for project "production line" including development, build servers, and test servers.
10. Automate test and nightly build, and measure performance.
11. Establish criteria for committing of code to integration.
12. Maintain integrity of configuration management, by using a checklist for Work Product Evaluation and execute it by the end of each sprint.

Conclusion

This paper presented a some practical advice for agile projects on additional activities to adopt particularly in larger or distributed projects.

Our recommendation to the Agile community is to extend agile methods inspired from an understanding of the mandatory goals and expected practices for CMMI level 2 and 3. These practices make good sense, and you could argue that it has always inherently been expected as part of your agile method. In general the CMMI model provides a good understanding what practices to consider – but you will have to adopt it to your context, and find agile implementations for the practices.

When projects grow, we believe you need more discipline. We have described how a more disciplined sprint zero, risk management, and various checklists with minimal effort can bring

you slightly more discipline into your project – and we believe that doing so will bring success to larger or distributed agile projects.

References

[1] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Implementation Guide*: Addison-Wesley, 2006.

[2] J. Sutherland, C.R. Jakobsen and K.A. Johnson, "CMMI and Scrum - a magic potion for code warriors" in proceedings for Agile 2007

[3] M. K. Kulpa and K. A. Johnson, *Interpreting the CMMI: A Process Improvement Approach, Second Edition*. Boca Raton: Auerbach Publications, 2008

Chapter 5: Scrum Metrics

From the beginning, Scrum was designed to provide transparency to both team members and those outside the team. The formation of the Scrum Alliance and the biannual Scrum Gatherings provides a forum for the Scrum community to further develop and refine approaches for management reporting.

A typical Scrum maintains a Scrum board showing columns of user stories, development tasks relating to each story, and the state of each development task and tests associated with each story. When a task is started a card is moved across the board into an open column. When code is complete it is move to the verification column and when fully tested is moved to a done column. An updated Scrum burndown chart along with a priotized list of impediments is often posted as well. Some sites even have Lava lamps that change color depending on the state of the build or state of the Sprint.

A manager can walk by a Scrum board and see the state of the team in a few seconds. If an impediments list is posted, a manager can add relevant items to his or her work list. This eliminates the need for most status reporting, particularly if critical information is put online on a web page, a wiki, or a reporting tool.

Managing and tracking costs to evaluate actual Return on Investment (ROI) can provide an additional feedback loop. Earned Value Management (EVM) may be of interest to management. Sulaiman, Barton, and Blackburn [86] provide detailed calculations for EVM and evaluation of its effectiveness in an Agile 2006 research paper.

Reporting Scrum Project Progress to Executive Management through Metrics

Brent Barton, Ken Schwaber, Dan Rawsthorne
Contributors: Francois Beauregard, Bill McMichael, Jean McAuliffe, Victor Szalvay
Scrum Alliance, 2005

Introduction

The interest in Agile software methodologies is not surprising. Agile methods are presenting an opportunity to develop software better and this is being noticed in the business community. Scrum is particularly of interest partly because of its ROI focus and quick implementation. While the efforts of innovators and early adopters have helped us assert that Agile is better than traditional methods, improving the reporting capability would help. Even better would be able to report project progress to executive management in a more compelling way. At a Scrum Gathering, white papers were submitted and discussed. This is a summary of those discussions and the integration of the contributions of many people. Visibility into project progress and project "health" is a consistent theme executive management desires.

Transparency into Projects

Executive Management needs transparency into all operations by viewing important indicators quickly: This is especially true of software projects. They want no surprises because in software a surprise is rarely a pleasant one. It is worth mentioning, however, that bad things do happen; executives know this and so does everyone else. It is always a surprise the first time one hears bad news. In contrast, the kind of surprise executives hate the most, have significant impact and were known much earlier than when they were finally informed. The negative emotional response to the surprise is reinforced by the realization that decisions were made on faulty information and this was preventable.

There are many techniques and practices for assessing the progress and probable success of projects. Scrum provides four simple and effective artifacts for managing and monitoring project performance: Product Backlog, Product Burndown, Sprint Backlog and Sprint Burndown. Building on these, we are integrating a Functional Work Breakdown Structure and a technique for measuring Earned Business Value.

Stakeholders and executives often have particular interest in certain areas of projects. The grouping nature of a Work Breakdown Structure (WBS) affords the opportunity to present progress at a mid-level: not a single view like a burndown and not at a detail level like a backlog. By combining a WBS, transparency can be attained quickly with a few simple, graphical reports on an executive dashboard

Executive Dashboard

The Executive Dashboard presented here is easily read, interpreted and provides the ability to reference additional material if desired (see Figure 2: Executive Dashboard).

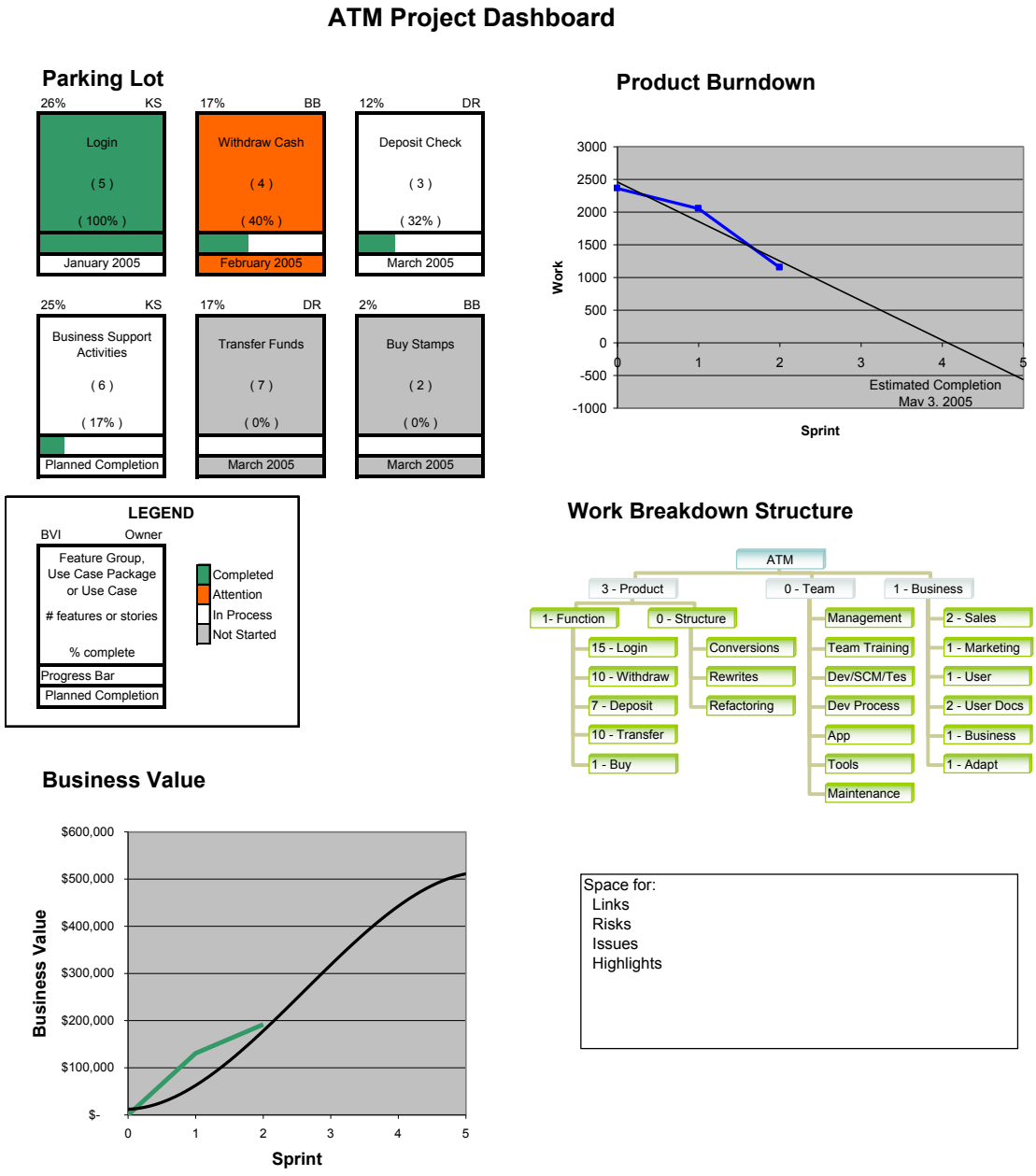


Figure 15: Executive Dashboard

The contents of this dashboard report include:

Parking Lot: This is a pictorial that statuses groups of features or use cases. This has been adopted from reports found in Feature Driven Development (FDD). With the addition of a Business Value Index (described later), one can see the progress and value of this area to the business. At a glance, the colors show where progress is made, areas of concern are and items not started. The BVI represents the total value of the project and the owner's initials describe who is responsible for the groups. The legend is included (see **Figure 3: Parking Lot**).

Figure 16: Parking Lot

Product Burndown: The burndown in work budgeted and planned compared as decreased by work completed across time. Based upon this, an estimated completion date can be determined as the trend line crosses the x-axis (see **Figure 4: Product Burndown**).
bbb
Figure 17: Product Burndown

Earned Business Value Graph: This presents the Business Value earned compared to the Planned Business Value. Variance can be quickly estimated from the graph to assess the correct prioritization and progress of the project Figure 5: Business Value Burnup).

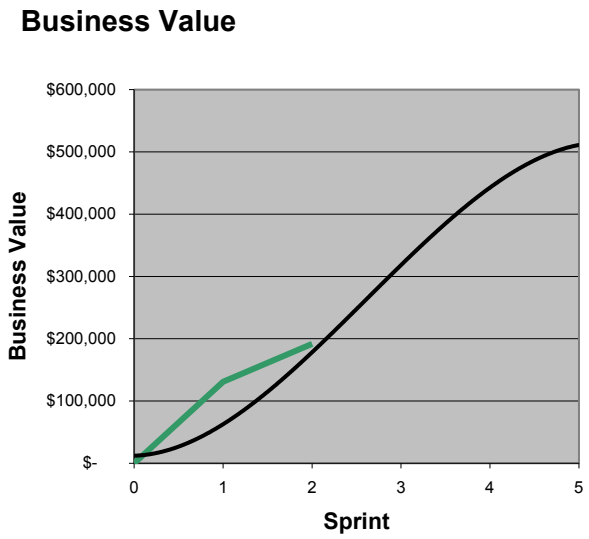


Figure 18: Business Value Burnup

Graphical Work Breakdown Structure: This visual representation provides a concise, high-level presentation of the project work items (see Figure 6: Functional Work Breakdown Structure). Space for links, highlights, issues and risks. Every project and customer has its own specific needs. This space is intended for a few bullet points.

Work Breakdown Structure

Dan Rawsthorne introduced a functional Work Breakdown Structure which provides us a structure for reporting key areas within a project and also measuring Earned Business Value. A Work Breakdown Structure provides "'A deliverable-oriented grouping of project elements which organizes and defines the total scope of the project.' [87]
b
Many think of Gantt charts and Microsoft Project Plans when they hear the term Work Breakdown Structure. This visually appealing format allows anyone to quickly see the salient work required to accomplish the project (Project at a Glance). This sample software project's WBS looks like the following, representing a fictitious ATM development project (see Figure 6: Functional Work Breakdown Structure).

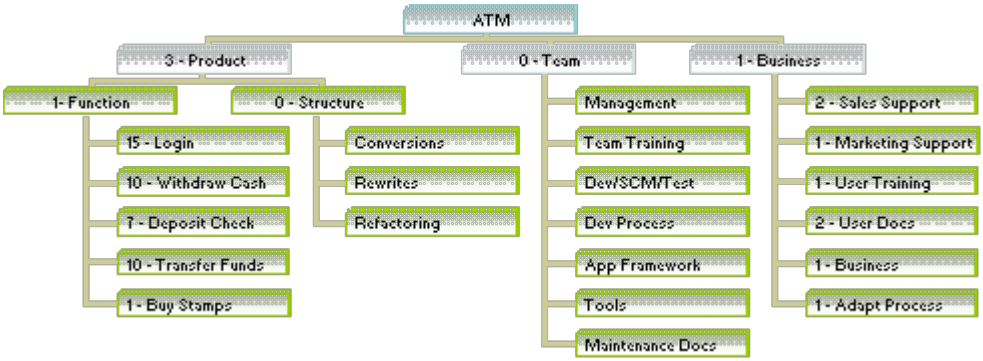


Figure 19: Functional Work Breakdown Structure

Each of bottom nodes in the Functional leg is a use case. A use case is not required but scenarios of use cases and stories align well and help produce useful software. Using other forms of requirements does not invalidate this structure.

Notice the numbers in the nodes that represent tangible things that can be valued. The other items are necessary to deliver the required results but do not have direct business value.

Earned Business Value

In order to represent the Earned Business Value (EBV) of a project and its components, an additive weight needs to be assigned. Total Business Value is determined by some ROI calculation or equivalent. Business Value becomes earned only when the items are done. In Scrum terms this means it is an "increment of potentially shippable functionality." Thus, only items of direct business value, such as functionality and training should be assigned weights other than zero. The other items are the cost of doing business. By calling them "orphans" they need to be adopted by items that do have value (Note: This is useful because it addresses total cost, not just cost-per-feature of a project and makes visible the cost of doing business in software. Also, the software team is reminded the difference between important work and business value of the output).

Detailed calculation of EBV is was published in the Agile 2006 proceedings . Here, only a brief overview is provided for one calculation [86]. In order to apply Business Value (BV) to a project, we need to calculate the Business Value Indexes. The Business Value Index (BVI) of the entire ATM project equals 1. For each level in the WBS, the index is 1: This is an intermediate value that will be used to calculate the BVI. To calculate the BVI of subsequent levels, you must also multiply the BVI of the all nodes above the level you are considering.

The next part is a bit more complex but the pattern is easy once you understand it. Remember, the BVI of the next level is 1*1=1, the index of the Project times the index of the next lower level. Because the functional leg has a weight of 3 and the business leg has a weight of 1, the sum of the additive weights of this level is 3+1=4. Thus, the Index of the functional leg is 3/4 and the business leg 1/4: This was obtained by dividing the weight of the leg by the sum of the weights on this leg. This is why additive weights must be used. So in order to calculate the BVI,

multiply the index of the target leg by the index of all the legs above it. Since the only leg above the functional leg is the ATM project, 3/4 * 1 = 3/4. the weight of the leg by the sum of the weights of all legs. Thus the BVI for the functional leg is 3/4 and for the business leg is 1/4.

Using the same method, the BVI of the login use case is 1 * 3/4 * 1 * 15/43 = 45/172. If the Total Business Value of the project is $500,000, then the Earned Business Value (EBV) realized by completing the Login Use case would be $130,814.

The Underlying Data

Managing and reporting effectively is a lot of work. The validity of the reports is only as good as the validity of the data. Figure 7: WBS in Product Backlog Format captures the WBS and calculates the BVI for each level. Note that each use case has been broken into stories (scenarios) and the weights and BVI have been calculated as well. For each item marked as done, the EBV is calculated. Notice the bottom row is done but there is no EBV because this represents a cost of doing business.

Area	Sub-Level	Use Cases	Value	BVI	Stories or Features	Value	BVI	Estimate	Done	Sprint	EBV
Product	Function	Login	15	26.2%	determine postconditions	0	0%	8	1	1	$ -
Product	Function	Login			Determine Main Success Scenario	0	0%	16	1	1	$ -
Product	Function	Login			Code up Main Success Scenario	10	20%	80	1	1	$ 100,626
Product	Function	Login			Extensions	0	0%	30	1	1	$ -
Product	Function	Login			Code Up "3 Strikes and You're Out"	3	6%	16	1	1	$ 30,188
Product	Function	Withdraw Cash	10	17.4%	determine postconditions	0	0%	12	1	2	$ -
Product	Function	Withdraw Cash			Determine Main Success Scenario	0	0%	20	1	2	$ -
Product	Function	Withdraw Cash			Code up Main Success Scenario	7	12%	120	1	2	$ 61,047
Product	Function	Withdraw Cash			Extensions	0	0%	30	0	2	$ -
Product	Function	Withdraw Cash			Code Up "Quick Cash Options"	3	5%	20	0	2	$ -
Product	Function	Deposit Check	7	12.2%	determine postconditions	0	0%	20	1	2	$ -
Product	Function	Deposit Check			Determine Main Success Scenario	0	0%	20	1	2	$ -
Product	Function	Deposit Check			Code up Main Success Scenario	10	10%	100	0	2	$ -
Product	Function	Deposit Check			Extensions	0	0%	30	1	2	$ -
Product	Function	Deposit Check			Code Up "Deposit Foreign Currency"	2	2%	50	0	2	$ -
Product	Function	Transfer Funds	10	17.4%				300	0	3	$ -
Product	Function	Buy Stamps	1	1.7%				100	0	3	$ -
Product	Structure	Conversions		0.0%				200	0		
Product	Structure	Rewrites		0.0%				300	0		
Product	Structure	Refactoring		0.0%				250	0		
Team	Team	Management		0.0%				120	0		
Team	Team	Team Training		0.0%				40	1	1	

Figure 20: WBS in Product Backlog Format

So, how does this information produce a dashboard?

Putting the Data into the Report

Sprint	Work Left
0	2362
1	2052
2	1152
3	
4	
5	

Figure 21: Product Burndown

The Product Burndown is automatic, using the columns, Estimate, Done and Sprint. Rather than complete automation, the auto-filter feature in Excel is applied for each Sprint and this can then be easily tabulated into this table (see Figure 7: Product Burndown). This is found in the header section of the Product Backlog in the attached spreadsheet. Using the subtotal function in Excel provides quick calculations from filtered data. Applying no filter

calculates the Work Left before the start of Sprint 1. Sprint 1 was found by using a custom filter where Sprint "is not equal to 1". Sprint 2 was found by applying the filter where Sprint "is greater than 1." As long as any items that are dropped form an active Sprint is reflected in the Product Backlog as planned for the next Sprint (unless the Product Owner changes that to a Sprint further out).

Creating the Parking Lot

The Parking Lot is created using the filters as well. For the "Deposit Check" use case, filter on Deposit Check in the use case column. The number of stories equals 5 so this will go in that middle of the diagram (see Figure 8: Deposit Check Progress Indicator). The total estimated time to completion is 220 and the total estimate of time where "Done" is filtered to 1 is 70. Thus, the percentage complete is 70/220 ~32%. Using the elements in the "Parking Lot Construction" tab in the spreadsheet, you can copy and paste the progress bar, so it graphically approximates this completion value. Since Scrum uses forward looking estimates, it is possible for a use case or feature group to show negative progress compared to a previous report as new information causes estimates to increase. The date is formatted in month and year but can be reformatted to reflect the lengths of iterations.

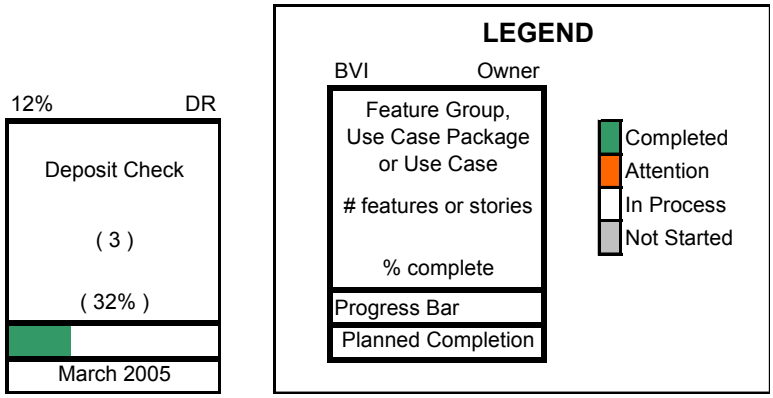


Figure 22: Deposit Check Progress Indicator

Earned Business Value

Earned Business Value is easily calculated by filtering on the Done field for a value of 1. Then filter on the Sprints up to and including the one being calculated using the custom filter "is less than or equal to." This yields the data in the EBV column for each Sprint (see Figure 9: Earned Business Value Data). This is located in the "EBV" tab in the accompanying spreadsheet. Note the Planned Business Value is calculated initially. Value is typically realized using an 'S' curve. The sample here has only 5 data points so the smoothing feature was used for the Planned Business Value. If the project is highly emergent (little planning beyond the current Sprint) the planned business value will only be one row ahead of the earned business value.

Conclusions

By taking advantage of the visibility that Agile methodologies provide, we can deliver meaningful information all the way up to Executive Management using graphical representations. The addition of a functional work breakdown structure provides the ability to view the project at multiple levels of granularity.

Sprint	Planned Business Value	Earned Business Value
0	$ -	$ -
1	$ 100,000	$ 130,814
2	$ 150,000	$ 191,861
3	$ 300,000	
4	$ 475,000	
5	$ 500,000	

Note:
This column represents skewed data because of graph smoothing. Because there are only 5 data points, a standard S-shaped Business Value is not represented well without smoothing

Figure 23: Earned Business Value Data

Notes

Many other metrics can be derived from the core data, including early warning indicators and schedule variances.

How weights are applied so they are additive can be debated or can be simple. This should be coordinated with the business and financial people to use their calculators to help derive this.

Chapter 6: Scrum Tuning

There is only one Scrum and one set of core processes. The inspect and adapt rules of Scrum allow it to be fine tuned for any organization. However, one of the goals of the first Scrum was to multiple productivity by an order of magnitude. The Smalltalk development environment created by the first Scrum at Easel Corporation had a money back guarantee to double productivity the first month of use. This may be the only money back guarantee of its type seen in the software industry.

There are some leaders in the Agile community that view productivity increases as marginal - zero to 25%. They do Agile development to improve quality and the lifestyle of the development team. While these goals are laudable, they are a prescription for getting your development team outsourced. There is a Scrum team in St. Petersburg, Russia, that can give you a velocity of seven times a typical waterfall team and developer salaries are much lower than in the U.S. or Europe. Scrum was designed to enable teams in the U.S. and Europe to produce software at half the cost of a waterfall team in India, Dubai, Russia, or China. Keeping a job is the first step to job satisfaction.

One of the motivating factors in creating Scrum was the Borland Quattro Pro project documented in detail by James Coplien when he was at ATT Bell Labs [2]. Each developer on this project generated 1000 lines of production C++ code every week for three years. In comparison, productivity on the recent Microsoft Vista project was 1000 lines of code per developer per year. The Borland Quattro Project was 52 times as productive as the Microsoft Vista project measured by lines of code.

Mike Cohn, in his book on User Stories [69], documents a well running Scrum of 6 people that reimplemented an application with 1/10 of the resources of a team that had used traditional Waterfall practices to implement that same functionality. The first Scrum at Easel Corporation achieved the same level of productivity. More recently, the SirsiDynix project demonstrated that an outsourced, distributed Scrum on a large project with over one million lines of Java code could achieve about the same high level of productivity [28] as small collocated teams.

It is therefore surprising when consulting with the many companies using Scrum that some get little increase in productivity, some double productivity, some quadruple it, but few achieve the order of magnitude increase for which Scrum was designed. Just like Rugby teams, some companies implement well and win all the time. Others do not do as well. When you look under the hood you find that the implementation of play is done poorly for some teams.

In an attempt to illustrate good, better, and best implementations of Scrum, implementations of Scrum were classified into Type A, Type B, and Type C. People thought these were different Scrums with different core practices. The core practices are the same. However, the style of some implementations are better than others with improved velocity and quality achievements. We wanted to analyze them so that Scrum practitioners could see the effect of rigorously implementing the basic Scrum practices and doing it in ways that radically improved development. This is Scrum tuning, or ways of implementing the core practices of Scrum at

different levels of rigor and adding some special sauce that takes the Scrum implementation from "good" to "great."

A better way to describe the evolution of Scrum styles is to show how focus on impediments in Scrum implementations paved the way for more rigorous implementations of Scrum. Initially we were concerned about making a team successful, then it became apparent the making the product successful was a higher goal. Once that was achieved, we noticed the best product doesn't always win. The company as a whole must inspect and adapt and become Agile along with the software developers.

Type B Scrum Continuous Flow: Advancing the State of the Art[3]

Jeff Sutherland, Ph.D.
Patientkeeper, Inc., Brighton, MA, 2005.

Introduction

Scrum is a process designed to add energy, focus, clarity, and transparency to software product development. It can increase speed of development, align individual and corporate objectives, create a culture driven by performance, support shareholder value creation, achieve consistent communication of performance at all levels, and enhance individual development and quality of life. Scrum adds value to the working life of software developers, while demonstrating to management that radically increased throughput is possible and sustainable over time.

Background on Scrum

In 1983 I began work on better software development processes at Mid-Continent Computer Services in Denver doing data processing for over 150 banks across the United States. This work was continued at Saddlebrook Corporation, Graphael, Inc., and Object Databases on the MIT campus in Cambridge, MA during 1986-1993. The goal was to deliver better software sooner while improving the working environment for developers. Software teams are often late and over budget and developers are usually blamed for failure. They are often punished and forced to endure "death marches" where they regularly burn out and quit as quality deteriorates, morale degenerates, and people are forced to work harder to produce less. Professor Peter Senge of MIT views this phenomenon as generic to U.S. management whose leadership is dedicated to mediocrity and destroys the spirit of the people [88].

In fact, as extensive data shows from an advanced Scrum implementation at PatientKeeper, Inc. (see later chapter), late projects are usually the result of management failure. Resources are continually diverted to non-critical activities instead of being fully allocated to projects, and corporate waste in the form of unnecessary meetings and beaurocracy is rampant. Management's failure to track and remove these corporate impediments causes productivity loss. A recent CIO Magazine survey showed that 96% of CIOs in the United States have no information on project status during the middle of a project. Flying blind leads to regular project crashes. Lack of visibility inside projects, particularly waterfall projects, allows management to blame the developers instead of taking the consequences and learning from their actions.

Project success is only a preliminary objective of Scrum. Our goal is to create an environment where developers exceed management expectations and create new products faster than marketing, sales, and customers can absorb them. This enables a world-class Scrum development

[3] First published in Cutter Agile Project Management Advisory Service: Executive Update, 2006.

team that can focus on exceptionally high quality and usability while positioning their company to dominate a market.

After evolving concepts at four companies, the first Scrum was launched at Easel Corporation in 1993 with the help of Jeff McKenna and John Scumniotales, the first ScrumMaster. Results exceeded all expectations and the experience was life-changing for the development team. In 1995, Ken Schwaber observed Scrum in action and presented the first paper on Scrum at OOPSLA. Ken and I co-developed Scrum for industry-wide deployment using many companies as laboratories to enhance and scale Scrum. In 1999, Mike Beedle standardized Scrum as an organizational pattern at the Pattern Languages of Programming Design (PLoP) Conference and in 2001, the three of us co-authored the Manifesto for Agile Software Development along with 14 other software experts. Two Scrum books were written by Ken Schwaber and Mike Beedle and the Scrum Alliance was formed .

A recent innovation has been automating real-time data collection for Scrum activities to reduce administrative time required from developers and product managers, and to allow better reporting to senior management. A side effect of this automation has been (1) collapsing bug reporting and status of development tasks into one system, (2) real time reporting on thousands of tasks on dozens of projects, (3) enhanced ability for the Product Owner to steer the development team in the right direction by better prioritization of the Product Backlog, (4) providing developers automated tasks lists where they can more easily self-manage work, (5) radically increasing throughput for multiple development teams, and (6) reducing project tracking overhead to 60 seconds a day for developers and 10 minutes a day for a project manager.

A side effect of reducing data collection costs to near zero has been increased awareness and management insight through total transparency of project status. This allows development leaders and senior management to evaluate process bottlenecks and remove wasted effort. The relationship between project failures and the lack of enforcement of lean manufacturing and lean product development principles can be precisely demonstrated through Scrum project tracking data. Lean principles used at Toyota and other companies drive continuous quality improvement and can be used as a method to analyse project management failures. Rothfuss [89] reviewed PatientKeeper's system and reported that it achieved:

- Unprecedented transparency
- Companywide visibility
- Metrics driven decision making
- Peer pressure and pride in workmanship driving productivity

These qualities make Scrum useful for implementation of Lean practices. These practices are useful for open source development as well as ISV product development or internal IT initiatives. Scrum is moving towards more broad application, while becoming faster, better, cheaper, and more rewarding for customers, developers, and managers.

Improving Scrum Implementations

One of the key influences that led to the creation of the Agile development processes, particularly Scrum, was a paper on Japanese new product development by Takeuchi and Nonaka [1]. The authors presented a chart showing product development activities (requirements, analysis, design, implementation, testing, deployment) separated into siloed phases of work (Type A or waterfall), phases slightly overlapped (Type B), and all phases of development overlapping (Type C). The Japanese viewed Type A product development at NASA as an outmoded relay race type of process. Type B at Fuji-Xerox and Type C at Honda they envisioned as similar to a Scrum formation in Rugby where all team members joined arms and phases of product development were overlapping.

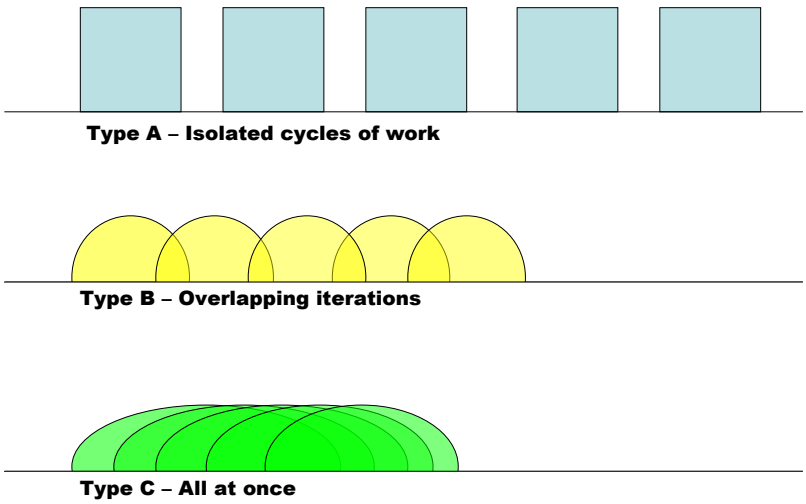


Figure 24: Type A, B, and C strategies for delivering product [11].

One can reinterpret this diagram at a higher level and apply it to different types of Scrum execution. Early Scrum implementations focus on helping the software development team to become more productive. Teams often do work within a timebox called a Sprint. If no work is done within the current Sprint to prepare for the next Sprint, time may be lost between iterations while reorganizing for the next Sprint (reset time in Lean manufacturing). If the Product Owner team in Scrum works on reprioritizing, specifying, and estimating (with developers) prior to the next Sprint Planning meeting, this time delay can be eliminated. By reducing reset time to zero, a Continuous Flow Scrum allows work to flow smoothly from Sprint to Sprint and can often double throughput. An All-at-Once Scrum can be envisioned as pipelining Sprints, i.e. running multiple overlapping Sprints through the same Scrum team. This is the type of process seen at PatientKeeper and at a few other software development companies.

Many companies, large and small, are experienced with Scrum and have the challenge of delivering a portfolio of products developed by multiple synchronized teams. While few are ready for a All-At-Once Scrum, most want to implement a Continous Flow Scrum. This involves introducing lean product development principles to guide the self-organization of a Scrum implementation in a company setting. There is also value in understanding and implementing some principles from lean manufacturing, lean logistics, and lean supplier management as lean is

a system of management of an entire organization, more than a collection of techniques and practices. Many of these practices are directly useful for software development [8].

Enhancing Scrum Implementations

The differences in a Team Scrum, Continuous Flow Scrum, and an All-At-Once Scrum have been well summarized by Ryan Martell [90] below. A major thrust of Scrum is examining, prioritizing, and removing impediments to shorten cycle time and deliver production software faster. Shorter delivery cycles means quicker time to market, faster evolution of the product, and increased knowledge gained by the product teams. Toyota has demonstrated that introducing lean manufacturing to shorten cycle time improves product quality and overwhelms competition. New product is introduced so fast and functionality increases and improves so quickly, the competition cannot keep up.

Table 1: Scrum characteristics [90].

	Team Scrum	Continuous Flow Scrum	All-At-Once Scrum
Iteration/Sprint Overlap	Down-time between Sprints/No overlap (insufficient Product Owner engagement)	Slight overlap (Planning/Prep)	Complete and Multiple
Level of Involvement	Development Team	Product Management Team	Entire Organization
Release	Every 4 - 6 Iterations	Every 2 – 4 Iterations	Every Iteration/Sprint
Iteration Lengths	Fixed	Fixed	Multiple Overlapping w/ varying lengths
Cycle Time for New Requests delivery	4-6 Months	2-3 Months	Monthly
Release per year	Total – 2 2 Major year patch as needed	Total – 4 Quarterly Major (4) patch as need	Total - 12 Weekly patch, (8) Monthly update (8) Quarterly Major (4)

Here, we focus on Continuous Flow Scrum which is the next step for companies that have mastered the basic Scrum process. Following the examples reviewed by Takeuchi and Nonaka [11] we set an audacious goal – cut release time in half, one of the prime directives of the Toyota Prius project. Average development time for design and implementation of a new automobile at Toyota was four years. The Prius team delivered in 15 months.

Aggressive goals put pressure on the system and processes must change. Engineering practices must improve. Software build processes must be automated. Testing must be started at the beginning as it cannot wait until development is done. Product reporting must be automated and visible to all in the company. A long list of impediments to progress emerge when cycle time is compressed. In Scrum, impediments are prioritized by the ScrumMaster and systematically eliminated with the assistance of the management team.

In Team Scrum, the focus is on team performance. Continuous Flow Scrum focuses on Program Management performance and product release (a combination of management and Product Owner activities). Continuous Flow Scrum does not exist if the company as a whole ignores (or is incapable of resolving) prioritized impediments raised by the ScrumMasters. The Program Management team must take responsibility for dealing with tough issues embedded in company cultures that sabotage performance.

Management Challenges for a Continuous Flow Scrum

The Japanese use the terms Muda, Muri, and Mura when applying lean manufacturing principles. Muda is waste. All waste, including every step that does not add customer value, must be eliminated. Muri is overburdening people, a process, or a system. Slack time must be introduced into a system to eliminate bottlenecks. Mura is unevenness or undue variation in process or product. Process flow must be smoothed out. These practices follow directly from queuing theory.

Many companies find it difficult to eliminate nonproductive assets, outmoded processes, or unnecessary hierarchy. Bloated artifacts, heavyweight project plans, and people who are not adding direct customer value need to be moved out of the way. The root of these problems [91] derives from the accounting system introduced by Sloan in the 1920's to measure success. Cost Accounting drives device optimization at the local level which suboptimizes global performance. Higher utilization of resources (people and machines) becomes an end in itself leading to higher production of inventory, rather than optimizing for sold products. Management forces people to work harder and harder to produce less and less

The Toyota process minimizes inventory and assumes process change occurs on a daily basis and these changes may affect the entire company. The Director of the Toyota Training Institute points out that the practice of "Kaizen mind" assumes that workers will become better at what they do every day [92]. Failure to do so is a corporate crisis and corporate cultures that are impervious to change cannot implement lean manufacturing practices.

Muri and Mura are the most insidious and difficult process burdens to eliminate. Old style management tries to load up a development team 110%. This makes them run slower, just as trying to run a computer process over 80% starts to slow it down, eventually causing the system to hang up. This Muri overload causes Mura (disruptions in flow) by generating process bottlenecks.

The Japanese have moved away from push type processes to pull type processes. When a worker needs to start the next step in a process, requirements and resources are provided just in time. The worker "pulls" these requirements and resources only when needed, allowing workflow to move at a steady pace. Scrum is designed to take advantage of lean manufacturing processes and a Continuous Flow Scrum can more than double productivity of Team Scrum.

Summary

During 2002-2007, the Scrum community of practice produced over 12000 Certified ScrumMasters in an introductory course. While these are only ScrumMasters in training, some of them have been able to transform entire companies. For example, Yahoo! has gone from one Certified ScrumMaster in 2005 to over 100 Scrum teams in 2007. For each Certified ScrumMaster, there appear to over 10 Scrum teams active on projects. These tens of thousands of projects are providing feedback to the Scrum Community through Scrumdevelopment@yahoogroups.com, biannual Scrum Gatherings, and an increasing number of research and experience publications.

Most of the leading software development companies worldwide have adopted Scrum as a development processes [93]. Many of those who have successfully executed Team Scrums for a year or more are ready to move to a Continuous Flow Scrum. This requires a corporate culture change to eliminate all processes that do not add immediate customer value. Management has to move to a facilitative leadership style and break down traditional hierarchies.

The Toyota consulting group recently selected the most Agile company it could find in the United States. In six months they improved productivity by 80%. Scrum has repeatedly demonstrated it can do the same or better. In 2006, a CMMI Level 5 company introduced Scrum as a process improvement intiative. Defects were reduced 40%, planning costs were reduced 80%, and overall productivity was up 100% [94]. They now bid Scrum projects at half the cost of waterfall projects. Winners may put their competition out of business just as Toyota is taking the lead away from General Motors.

Postscript

This paper provided a brief overview of a Continuous Flow Scrum when it was originally published. A more profound insight into these issues can be found through study of set-based concurrent engineering [56] and the theory of constraints [95].

When this paper was originally written for the Cutter Agile Project Management Advisory Service, the editors asked that it be watered down. They did not want me to state that Toyota would overtake General Motors. It had forcefully made the statement that Continous Flow Scrum is designed to create the Toyota effect of four times productivity and 12 times better quality than competitors. This level of performance will put the competition out of business. It is inevitable. The only question is one of timing and one only has to look at Toyota's impact on General Motors. Little did I know how quickly that problem would become of immediate importance.

General Motors in recent years has improved the performance of employees such that the time spent on adding real value to the customer has moved from a traditional 20% up close to Toyota's 80%. Nevertheless, as Forbes columnist Marilyn Cohen observes:

"The domestic automakers are in a stew, with General Motors in the deepest. Nevertheless, GM, the largest company in what seems to be a dying U.S. industry, may get a new life. At least that's the possibility held out ... (for) an alliance between GM's Chief Richard Wagoner, and

Renault/Nissan Motor's, Carlos Ghosn… Then long-suffering GM bondholders won't be exposed to a Chapter 11 filing, which is the fate many investors mentally assigned the company not long ago [96]."

Such is the Toyota Way side-effect. Of even more interest, the strategic target for Toyota and some other Japanese car manufacturers is not General Motors, it is to eliminate the internal combustion engine. The lack of Agility and the ability to innovate in an environment where leaders are totally changing the playing field will ultimately force the creative destruction and acquisition of assets of the lame and the weak.

Type C All-At-Once Scrum: Creating a Scrum Company

Jeff Sutherland, Ph.D.
Patientkeeper, Inc., Newton, MA, US

"This is a very important paper! It lays out a masterful series of business process innovations that desperately need to be emulated by most organizations." Tom Poppendieck

Abstract

Scrum was invented to rapidly drive innovative new product to market. Six month releases used to be a reasonable time from for an enterprise system. Now it is three months for a major new release, one month for upgrades, and one week for maintenance releases. The Scrum development process was designed to enhance productivity and reduce time to market for new product. In this paper, one of the inventors of Scrum goes back to Scrum basics and designs All-At-Once Scrum using multiple overlapping Sprints within the same Scrum teams. This methodology delivers increasing application functionality to market at a pace that overwhelms competitors. To capture dominant market share requires senior management participation in a MetaScrum for release planning, variable length Sprints, overlapping Sprints for a single team, pre-staging Product Backlog, daily Scrum of Scrums meetings, and automation and integration of Product Backlog and Sprint Backlog with real-time reporting. A practical example of All-At-Once Scrum describes how mobile/wireless product teams implemented Scrum process automation beginning in 2000 and achieved hyperproductive revenue growth by displacing dominant vendors in 2007. Administrative overhead for over 45 enterprise product releases a year is less than 60 seconds a day per developer and less than 10 minutes a day for a Project Manager. While All-At-Once Scrum is not for beginners, this professional implementation of Scrum companywide is faster, better, and cooler.

1. Scrum Evolution

Evolution occurs in dynamic response to environmental demands. Now that the Scrum community has 15000 Certified ScrumMasters and hundreds of thousands of projects under their belt, retrospection can help guide future activities. In particular, what did you do yesterday that worked (Scrum theory), what makes sense to do tomorrow (Scrum evolution), and what is blocking the way (Scrum dogma) is worthy of analysis.

One of the key influences sparking the creation of the Scrum Agile development process was a Harvard Business Review paper on Japanese new product development by Takeuchi and Nonaka [1]. A key component of their presentation was a chart showing product development separated into silo's (Type A), phases slightly overlapped (Type B), and all phases of development overlapping (Type C). The Japanese viewed Type A product development as an outmoded relay race type of process. Type B they thought was similar to Sashimi because slices of work overlapped requiring collaboration between phases. Type C they envisioned as Scrum where all phases of product development were done simultaneously. Scrum is a Rugby formation and they viewed an "all-at-once" process as similar to a Rugby team moving down the field passing the ball back and forth to one another.

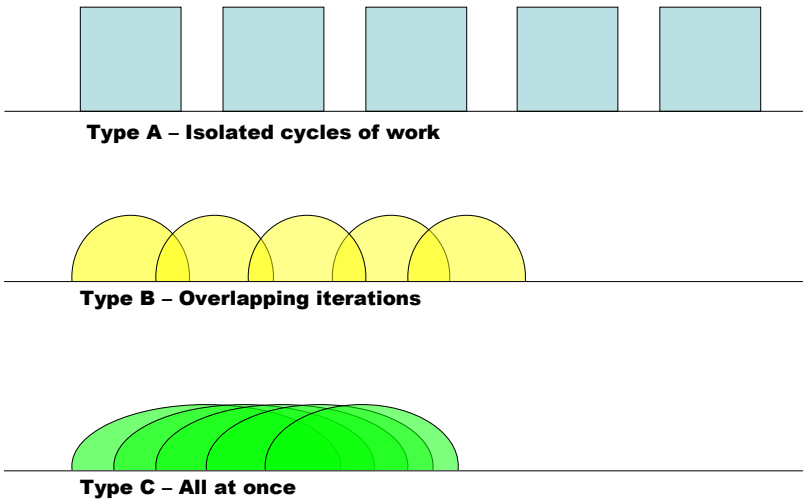


Figure 25: Type A, B, and C strategies for delivering product [11].

After discussing the notion of various types of Scrum with Certified ScrumMasters in Scrum Alliance workshops and with development teams at Google, Microsoft, Yahoo, Arriba, Adobe, GE Healthcare, and other companies, it appeared that the chart above can be applied to a higher level of thinking about three styles of Scrum implementation.

In early Scrum implementations (called here a Team Scrum), all development occurs within the timebox of a Scrum iteration called a Sprint. A side effect of this approach is downtime between iterations when reorganizing for the next Sprint. Well executed Sprints can double productivity and repeatedly deliver projects on time, within budget, with functionality precisely targeted to end-user demands.

By adding product definition tasks for the next Sprint into the current Sprint, a Continuous Flow Scrum allows work to flow smoothly from Sprint to Sprint. Product backlog requirements for the next Sprint are developed in the current Sprint. While this requires that a development team allocate a small amount of time from the current Sprint to help the Product Owner estimate the Product Backlog for subsequent Sprints, it can enable high performance development organizations to deliver more working product than sales, marketing, or customers can absorb. By eliminated the development bottleneck the company can adopt new strategies and create new products that were previously impossible to deliver. Most companies today have to implement a Continuous Flow Scrum out of necessity to deliver a continous set of product portfolio releases to the market. However, high performance of a Continuous Flow Scrum requires rigorous implementation of lean practices.

All-At-Once Scrum can be envisioned as pipelining Sprints by running multiple overlapping Sprints through the same set of Scrum teams. This requires experienced Scrum teams, well designed product architecture, and automation of Product and Sprint Backlogs. Throughput can be enhanced to deliver dozens of new releases of enterprise software annually. A MetaScrum is

implemented to create Agile product release strategies. On a weekly basis, companies can alter their direction to deal with competitive threats and market changes. Competitors can be overwhelmed and market dominance achieved.

Takeuchi and Nonaka observed collapsing phases of product development improved innovation, throughput, time to market, and product acceptance. As market pressures have evolved and changed, it is possible to collapse Scrum Sprints to create more business opportunity. Market domination is the goal of the All-At-Once Scrum.

2. Scrum Evolution in Practice

The evolution of Scrum in five companies from 1993-2001 has been described previously [17, 39]. Here we focus on continued evolution of Scrum theory using PatientKeeper, Inc., as a test bed. During 2001-2005 we automated a solution for a Continuous Flow Scrum. This eliminated lost time and productivity between Sprints and, as observed previously at Easel Corporation in 1994, significantly increased throughput compared to completing work only within the Sprint time box for which it is defined.

In addition, we solved the problem of multiple projects pipelined through the same team (or set of teams) and have been running an All-At-Once Scrum for over six years. This required careful automation of the sprint backlog with improved tools and metrics in order to maintain team focus. Daily build processes and automated regression testing was significantly enhanced. Our approach to Quality Assurance (QA) was modified to provide a small QA team for each of four to six overlapping production software releases. Pair programming was used selectively and team programming was common where many programmers worked together around a table for the entire day.

The result has been delivery of production code to a new set of enterprise customers for every Sprint with maintenance Sprints weekly, customer enhancement Sprints monthly, and new application releases quarterly. By 2004 more than 45 enterprise level releases of PatientKeeper production software were completed, installed, and brought live at customer sites. Many of PatientKeeper's customers are large multi-hospital systems like Partners (Massachusetts General and Brigham and Womens Hospitals) in Boston, Johns Hopkins in Baltimore, and Duke University Health System in North Carolina.

Recently, PatientKeeper broke a record in deployment time. For HCA, the largest hospital system in the U.S., PatientKeeper brought 12 hospital deployments into full production in seven months with new physician and administrative desktop and PDA applications. This was actually slow by PatientKeeper standards, yet was a historical record for HCA.

These large clients provide an excellent test bed for scalability of an All-At-Once Scrum. They require a high level of product adoption by difficult and discriminating physician users, support for large multi-institution wireless networks, integration with many clinical and financial systems at sites with diverse IT infrastructures, and thorough testing and certification by the customer.

2. The First Scrums – Team Scrum and Continuous Flow Scrum

Early Team Scrums were useful for education and training on the pace of Scrum and particularly suited to new Scrum teams. However, it creates a loss of time between Sprints where the team is reorganizing for the next Sprint and did not adequately address enterprise scaling issues.

At Easel Corporation in 1993 we initially applied Team Scrum to software development teams when we built the first object-oriented design and analysis (OOAD) tool that incorporated round-trip engineering from design to code and back again In a Smalltalk development environment [21]. Code was generated from a graphic design tool and any changes to the code from the Smalltalk integrated development environment (IDE) were immediately reflected back into design. There were six Sprints for the first product release and the gap between Sprints took at least a week and sometimes two weeks. As a result, we could only do 9 Sprints a year, losing 25% of our productivity as compared to potentially running 12 Sprints per year. This was viewed as a serious impediment by the Product Owner and management.

This loss of time needed to be removed because survival of the company depended on delivery of an innovative product as early to market as possible. Each month of delay cost millions of dollars of lost revenue and gave the competition the opportunity to overtake us. For example, the Product Manager at Rational Rose was regularly showing us demos of support for roundtrip engineering as we were struggling to be first to market with the first roundtrip engineering object-oriented design tool.

In addition to loss of productivity between Sprints in a Team Scrum, it took time during the Sprint for the developers to get enough clarity about the user requirements to start coding. It may be halfway through a Sprint before the developers understand the user experience well enough to implement a solution. This creates tension between the Product Owner and the Scrum Team concerning lack of understanding of what to do next, substantial slippage of features into subsequent Sprints, and dissatisfaction on the part of the Product Owner with delays in feature delivery. This churning phenomena can cut Sprint productivity in half, a huge impediment that needs to be remedied.

The need to start development with adequate functional specifications was observed by MacCormack [97] when he gathered extensive data on 29 Hewlett Packard software projects to assess development practices. One of the strongest productivity enhancers noted in his correlation analysis was completeness of the functional specification. While Agile specifications are designed to be just enough and no more, a product specification needs to be "ready" before it can be allowed into a Sprint. MacCormack showed the definition of "ready" has major performance implications.

Regarding the use of specifications, there was a significant relationship between the completeness of the functional specification and productivity. There was a weak relationship between the completeness of the detailed design specification and defect rate ($p = 0.078$). The former result suggests that developers are more productive to the degree that a complete functional specification exists prior to coding. This is intuitive, given that the functional

specification outlines the features that developers must complete. To the degree that these are stated up front, developers can focus solely on "executing" these features in code.

Agile developers use a minimum amount of documentation and do not require completeness of the specification to start a Scrum. McCormack found that completeness of the design specification was not correlated with enhanced productivity and only slightly reduced the defect rate which is consistent with Agile thinking. However, he found a strong correlation between adequate product specifications and productivity. This suggests that minimal functional specifications should be clear at the beginning of a Sprint and that design and technical specifications are best done within a Sprint.

MacCormack's multivariate analysis showed three primary factors that lowered defect rate (early prototype, design reviews, and integration or regression testing at code checkin) and two primary factors that increased productivity (early prototype and daily builds). Releasing a prototype to customers that is only 40% functionally complete increases productivity by 36% and adopting the practice of daily builds increases productivity by 93%. These were clearly the most effective Agile practices in the Hewlett Packard projects.

Incremental and early delivery of working software is at the core of the effectiveness of Agile processes. In addition, a functional specification that is complete enough for the next interation to allow developers to begin work without false starts will enhance feature delivery within Sprints and improve throughput. Despite the fact that the implementation phase is a small part of the overall cost of a software project, the biggest resource bottleneck on a software project typically occurs with a shortage of expert developers whose skills are not easily transferable. Constraint analysis shows mathematically that the biggest bottlenecks should be eliminated first [95] (just as in tuning of a computer system) and early delivery of a functional specification for a single increment helps eliminate the critical development resource bottleneck.

While a Continuous Flow Scrum can improve productivity with experienced Scrum teams, a Team Scrum with intervals between Sprints to assure the Product Backlog is "ready" may be the best way for a company to pilot Scrum, even though it may not be most efficient. It allows systematic application of the Scrum process with enough time to refine operations and regroup between Sprints. It also forces all-at-once type thinking when everything has to happen for a specific Sprint within the time box of that Sprint. Initially, the benefits in training may overwhelm the lost productivity. Without the ability to execute a Team Scrum well, it is not possible to effectively implement a more sophisticated process.

The benefits of Team Scrum are:

- Total focus on iteration in process
- Ease of implementation
- Developing and understanding the pace of Scrum
- Clearly defined iterations

The problems with Team Scrum were:

- Loss of time to market
- Disruption of pace of Scrum because of delays in understanding of the user experience

- Loss of productivity (and market share) due to resulting delays

Scrum has a primary focus on an impediment list managed by the ScrumMaster. By prioritizing this list which includes personal, team, and company issues, the ScrumMaster puts an intense focus on improving the Scrum implementation. When removing impediments most companies will find they need to go to a Continous Flow Scrum to maximize throughput.

3. Continuous Flow Scrum

The way to overcome loss of time to market with a Team Scrum is to insert tasks in a current Sprint that prestage work for a subsequent Sprint. A minimal specification of the user experience for a feature is defined and estimated prior to the Sprint where it is implemented. This allows Sprints to be executed continuously with the Sprint Backlog always full at the beginning of each Sprint. At the same time, it requires that a Scrum Team allocate resources to help the Product Owner estimate features for subsequent Sprints during the current Sprint.

A caveat is that Continuous Flow Scrum will not work in a company that has not implemented a sustainable development policy and process. That means that Scrum teams decide on what tasks can be implemented in a Sprint and who will implement them using a normal work week as the standard way to do business. Many companies using Scrum still have management trying to jam more work into increments than Scrum teams can deliver in an allotted time. This results in lack of team autonomy, excessive overtime, high defect rates, personnel burnout, and high employee turnover. This violates a fundamental principle of lean product develop and makes it impossible for a team to enter the high performance state for which Scrum was designed.

The key indicators that Scrum is working must be visible in a Team Scrum before moving to Continous Flow Scrum:

- Team autonomy – the Scrum team is (and feels) totally responsible for their product and no outside agency impacts the workplan of the team inside a Sprint.
- The Product Owner is part of the Scrum and affects product design and implementation within a Sprint without disrupting self-organization.
- Self-transcendence – individuals move beyond self-gratification to focus on team performance.
- Cross-fertilization – expertise is regularly shared across team members and no single person is a bottleneck.

Fully loading the development queue in a Type B Scrum at all times without building a sustainable pace of development will negatively impact morale. On complex development projects, it typically takes a new engineer six months to come up to full productivity. If turnover is 20%, you lose one quarter in hiring a new development and two quarters training them. Your development team productivity is down 15% from this alone. This personnel churn can cause development tasks to stop and start as specialized resources must be shifted to complete them, reducing productivity by another 15%. If morale drives the pace of development down further, you may cut productivity in half with Type B Sprints that are implemented too early.

Conversely, if Type A Sprints are running well, pre-staging functional specifications in the right way in a Type B Scrum will eliminate churn within a Sprint and downtime between Sprints. This has doubled productivity for experienced Scrum teams. In companies seeking to expand market share and dominant a market segment, this advantage is absolutely compelling.

In several companies using Scrum, management and staff stated that maximizing development throughput was not a priority. These companies invariably were having delivery problems and were outsourcing major pieces of their development. Outsourcing was not solving their delivery problems and in many cases was aggravating it. In the long run, this last to market approach is a losing strategy. If companies are not continually getting better, failure is just a matter of time as the competition is always improving.

3.1 Staging Functional Specifications for a Type B Sprint

A Type B Scrum accelerates throughput by keeping the Sprint backlog full and at times. This requires prestaging functional specifications of the product prior to the start of a Sprint. Maintaining the agility of the Scrum process requires a minimalist approach to functional specifications which is just enough, and no less than just enough.

A minimal amount of documentation for a product feature is typically a few pages and definitely not hundreds of pages. Just enough documentation so that engineers understand the user experience will suffice. This typically means screen shots, data requirements, workflow from screen to screen, and business logic that must be executed. The minimum documentation required to achieve Jacobsen's overview of an object-oriented analysis of the problem [98] is an excellent guideline even though it may be beyond the capability of some product management organizations. Fortunately, PatientKeeper had well educated physicians that served as Product Owners. While some of them had no formal training in software development, they learned quickly how to elaborate use cases in a way that defined the user experience for the physician when using PatientKeeper products. In addition, these physicians where action oriented and strongly resistant to analysis paralysis. They avoided time spent on excess documentation, making them excellent Agile developers by inclination.

Moving to a Type B Scrum requires analysis and design resources from the development team in order to help the Product Owner create functional specifications and pre-stage the Sprint backlog for the next sprint. In the worst case, this might require 25% of the Scrum resources during a sprint. However, it avoids the 25% lag time between sprints. So in the worst case you may break even on resource allocation.

The real gain from a Type B Scrum is having the Sprint backlog fully loaded at all times. A developer never wonders what to do next because the queue is always full. If the Sprint backlog is automated, team members simply logon at the beginning of the day and self manage the queue of work in front of them on a web page. The Product Owner and ScrumMaster are continuously working to assign items to a developer's queue and the developer decides how to order the work or, in some cases, will reassign it to a more appropriate team member. This radically increases throughput during a Sprint, often doubling it.

A relevant analogy is water flow through a garden hose. If the faucet is turned on and off, you disrupt flow and generate pressure surges in the line. These cause side effects and sometimes structural breakdown in water lines feeding the hose. Any structural breakdown will reduce productivity to zero for an extended period. Keeping the faucet turned on, even with reduced flow, may generate more throughput without negatively impacting upstream systems.

At the time of the PatientKeeper first major round of venture funding in 2000, I asked Ken Schwaber to help get a Type B Scrum launched immediately. Product Management owned the products and they were required to define the user experience of a feature before it could enter a Sprint backlog. More specifically, because of the demanding requirements of a physician user, the screen shots, the logic, the workflow between screens, and the data items required had to be defined. In addition, a prototype had to be tested with physician users and validated so that we knew conclusively that (1) the functionality was needed, and (2) the physicians would use it when it was implemented.

The requirement that a Product Owner provide a functional specification sufficient to clarify the user experience creates a positive dynamic tension between the Product Owners and the Scrum teams. The Product Owner cannot get a feature into the queue for development unless it is defined enough for the Scrum team to begin work immediately, either by building a technical design document or coding directly from the Product Management specification when possible.

At the same time, we gave the Product Owner resources in any Sprint to create a prototype of a new feature using very rapid mockup tools that would create screen interactions on a mobile device. In addition, the Product Owner had complete control over moving items in and out of the Sprint Backlog before and during the Sprint. This puts the Product Owner in the driver's seat. This is effective only if the Product Owner knows how to get to the destination, i.e. the right product specification is ready for the Scrum team to implement at the right time.

By holding Project Owners responsible for defining the user experience for a feature prior to the start of the Sprint, a rigorous process had to be introduced into Product Marketing at PatientKeeper. The process made it crystal clear that building new product began only when the Product Manager's job was done. At the same time, it was important to always have the development queue loaded. Management insisted that the development team not have downtime. The developers self-managed their work queue and the work queue was always full or a company emergency was declared, similar to the build process breaking. The only way Product Marketing could get something on the queue is was to complete their minimalist functional specification, just enough information in just the right format at just the right time.

A positive dynamic tension was created because the Product Owner (in this case, Product Marketing) always wants more product in a release and tries to jam features into the development queue. Developers always have more work than they have available time. In a Type B Scrum, it does not matter whether Product Marketing introduces new features into the queue in time or not, development productivity is not impeded. They just work on what is in the queue. Management is happy because they are always seeing features roll out. If the mix of features is

off, they hold the Product Owner responsible when the required functional specifications were not ready.

3.2 Product Owner as Part of the Scrum Team

The original Japanese view of a product development Scrum created a team that was totally responsible for the product [1]. In some companies, such as Individual in 1996, the Product Owner was at every Scrum meeting. In others, like the original Scrums at Easel Corporation in 1993-94, the Product Owner was on the road much of the week and was always at the Friday Scrum meetings [17, 39].

The Product Owner owns the business plan for the product, the functional specification for the product, the product backlog for the product, and prioritization of the product backlog. As a member of the Scrum s/he works side by side with the ScrumMaster to introduce product backlog items into a Sprint where they are broken down into tasks by the team for execution as Sprint backlog. At PatientKeeper, the Product Owner manages the movement of tasks in and out of the Sprint backlog in consultation with the ScrumMaster.

The linkage can be very tight between Product Owner and ScrumMaster with highly skilled people. For example, at PatientKeeper, the mobile device development team leader is the lead designer and coder on the team, the ScrumMaster, and one of three Product Owners in the company, reporting to both the VP of Marketing and the Director of Engineering. The two other Product Owners are responsible for clinical applications and financial applications on handheld and web devices. For clinical applications, the clinical Product Owner and the mobile device Product Owner are joined at the hip with the ScrumMasters. Together, they are totally responsible for the business plan, the product specification, and working day to day with engineers on product design.

After working as head of engineering in nine companies, I have found that the best way to think about Scrum responsibilities is to think of Scrum team as analogous to a high performance car in a rally race. The Product Owner is the navigator and the ScrumMaster is the driver. The team is the engine, the chassis, the drive train, and the wheels. The ScrumMaster follows the navigational directions of the Product Owner precisely and drives the car adroitly. The car and its occupants are totally responsible for winning the race. At the end of every Sprint, other players move in and can make modifications to improve the timing of the next Sprint.

New ScrumMasters tend to view this analogy as too prescriptive to a team that assigns its own responsibilities and self-organizes. Consider a football team. It self-organizes under the Coach where those best suited to roles, assume positions of quarterback, center, tight ends, and so forth. In the huddle, they may quickly make minor refinements to individual roles and responsibilities. However, when the ball is hiked, there is no discussion of what anyone is supposed to do. To be successful, they must know what they are to do and execute it quickly and professionally.

In some companies new to Scrum, engineers have claimed no one is responsible because there is not a “project manager.” If you look at project managers in technically driven companies, they

are usually at the mercy of the technical team. As a result, product management, and therefore product ownership, is weak. This compromises the effectiveness of Scrum and prevents a Scrum team from entering a hyperproductive state [25].

In a well run Scrum, particularly a Type B or C Scrum, the ScrumMaster must be able to drive the race car and the Product Owner must be able to direct the ScrumMaster to the destination in the timing necessary to win market share and attain profitability. Failure at either of these tasks leads to replacement of the appropriate person. Success means one or both go on to greater responsibilities. For those who lead hyperproductive Scrums, career advancement is rapid and they usually wind up as CTOs, CEOs of their own companies, or VPs of Engineering of larger companies within a few years. The ScrumMaster of the IDX Web Team left the Scrum team to lead the U.K. division of IDX and closed over $2B of new business within 3 years of leaving the Scrum. This is a example of a great ScrumMaster who learned how to own the business as well as the technology, side by side with the Product Owner.

3.3 Type B Scrum Can Enable a Hyperproductive Team

Giving the Product Owner control of the Sprint backlog along with strong accountability builds strong product managers. It also conditions the development team to move rapidly towards the goal without analysis paralysis. A combination of dynamic forces wins a race when a forceful driver is coupled to a high performance sports car or a high spirited stallion. The same phenomenon happens on sports teams when everyone understands the plays and can execute them immediately on demand. It allows the team to move up to a higher level of play where the basic moves are on autopilot and superlative moves are possible. The first Scrum began executing Type B Scrum as they mastered the process. They were able to enter the “zone” using this technique, where they could deliver functionality faster than the customers, the marketing team, or sales could absorb product. The feeling of power in a development team that can deliver more product than anyone can absorb is exhilarating.

Scrum was designed for this hyperproductive state, to get ordinary developers to function as a champion team. It only happens to about 10% of Scrums and it only starts to happen when the organization moves to a Type B Scrum. The doubling of throughput from a team that is already very productive results in an organizational breakthrough.

4. Type C Scrum

Scrum is an organizational pattern that is designed for an activity that is difficult to control because its predictability is limited [25]. It is useful in any context where the activity requires constant change in direction, unforeseen interaction with many participants, and the need to add new tasks as work unfolds. These factors were amplified at PatientKeeper when it received a $50M round of venture funding in 2000.

A decision was made to become a platform as well as application company with a software framework and open application programming interfaces (APIs) that would allow integration

with development partners on both back end servers and the front-end devices. A web services platform architecture was selected.

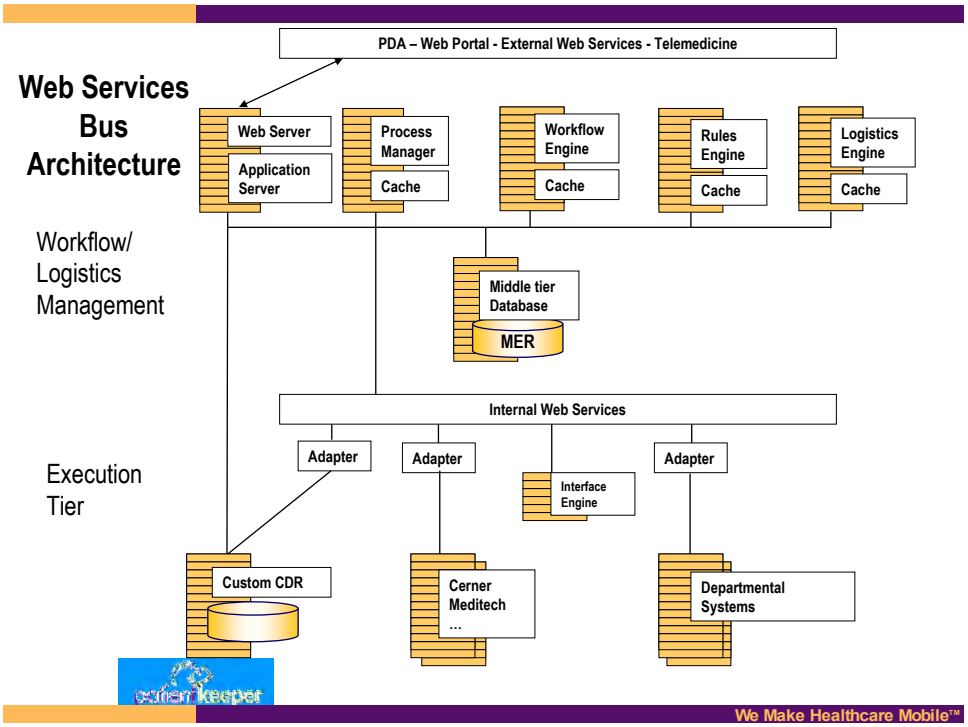


Figure 26: Patientkeeper platform architecture [99].

In addition to the server/network architecture based on Java and XML, a cross platform software framework on Palm and Pocket PC handheld devices was implemented in C/C++. This framework provided open APIs and a software development kit that allowed third party vendors and end users to tightly integrate their mobile applications with other applications already available on a handheld device.

The tight integration between software components required tight integration of software development teams internally at PatientKeeper and externally with partners and offshore developers. This, combined with time to market pressure and rapid growth of new client deployments in large enterprises, each demanding new increments of functionality along with 30-90 day installations, forced a new type of Scrum to be implemented at PatientKeeper.

4.1 Case Study Context

PatientKeeper builds a software platform that takes information from many clinical systems across multiple hospitals and clinics and presents it on an intuitive user interface to physicians using handheld devices and the web. The application software has a four tier architecture with four levels of data cache:

- Primary data source is a clinical data repository
- Data is forward cached in a mini-clinical data repository
- In-memory cache exists on a middle-ware server to improve performance
- On a handheld device, the data is stored locally

Software and data must be consistent across four tiers at all times. This forced PatientKeeper to go to totally integrated builds at all times to assure that software in all four tiers of the architecture worked consistently. Software testing has to validate that version of all architectural layers work together to provide consistent data to the end user.

The developer team working on this product was split into a backend integration team, a clinical repository team, a middleware server team, two PDA teams (Palm and Pocket PC) and a Web team. It was necessary to tightly couple these teams together to assure that all software was interoperable all the time.

4.2 Case Study Market Requirements

As an early-stage, venture funded company, PatientKeeper had to establish a new product offering in the rapidly growing mobile/wireless market. Early customers had to be implemented as quickly as possible with limited functionality. Subsequent customers needed to be installed as soon as possible with upgraded functionality. The imperative was to gain market share and achieve market dominance in a highly competitive environment. Speed to market needed to be used as a strategic weapon. Company viability and success demanded it.

The customer base rapidly evolved to 5-10 hospital systems to be installed each quarter. Each group of hospitals needed more extensive functionality in a rapidly growing portfolio of applications that included business partners with integrated back end clinical systems, portal vendors, and handheld device application vendors. A major release with new applications was required on a quarterly basis.

Customers consistent of large hospital systems with multiple installations, academic research institutions with integration into home-grown healthcare research and treatment systems, and medium size community hospitals with different requirements. The quarterly major release had to be supplemented with monthly minor releases to allow multiple new installs of similar clients to take place on a monthly basis. Finally, bugs and unanticipated implementation issues that had to be resolved to go live at new sites required maintenance releases every week or two.

4.3 Case Study Problem

The challenge for PatientKeeper quickly became how to simultaneously do weekly, monthly, and quarterly releases of a system that was tightly integrated across four architectural layers with six software development teams that needed to be tightly coupled to assure data and application

consistency across multiple back ends, diverse wireless networks, and multiple front end devices. Furthermore, each release had to be tested and certified across all levels of architecture and applications for deployment.

PatientKeeper started up as a Scrum company doing daily meetings. Type B Scrum had been implemented from the beginning with strong product owners required to produce functional specifications before any product backlog item could be transformed into tasks for a sprint backlog to be implemented in the next iteration. The challenge was to a automate project management to monitor thousands of tasks across dozens of releases a year without disrupting the Scrum process.

4.4 Case Study Forces

Resource constraints forced every developer to be focused 100% on building the system. ScrumMasters and team leaders spent the majority of their time designing and coding the system. Separate project leaders were not an option.

High caliber developers, many with doctoral degrees, did not want excessive administrative overhead. They felt that project management could be automated and taken to a new level of efficiency. The CTO of PatientKeeper was asked by the Scrum teams to organize a project management system that required less than 60 seconds per day of administrative time per developer and less than 10 minutes per day for a ScrumMaster to provide comprehensive reporting to management, the development team, and other areas of the company.

- Estimation was important. How were developers going to provide valid estimates and update them in less than sixty seconds a day?
- Planning and prioritizing takes time. How was this going to be accomplished without impeding development throughput?
- Architecture was critical for a platform company. How was it going to evolve using the Scrum process to provide flexibility, scalability, performance, reliability, and maintainability?
- Customer requirements in the form of use cases that could be rapidly transformed into deliverable code were essential. Who was going to do them and how would they be delivered?

4.5 Type C Scrum Solution

The Type C Scrum solution required several innovations that affected all parts of the company. In effect, the company had to become a Scrum company with all activities driven by an automated data system that reflected release planning and Sprint execution, as well as installation and support team and customer feedback.

- Team organization was changed

- Build process became more highly automated
- Regression testing automation improved significantly
- All data collection had to be automated
- New tools for data collection and reported had to be developed
- A MetaScrum had to be created to allow company leadership to manage multiple simultaneous product releases
- New reports had to be developed
- The company had to become totally transparent. All data was available to everyone in real time all the time.

4.5.1 Team Organization

Figure 27: Open space for Type C Scrum of Scrums daily meeting

The daily Scrum meeting quickly evolved into a daily Scrum of Scrum meetings. All members of the development team are present for 15 minutes meetings. Team leaders do most of the reporting:

- What did each of the six integrated teams complete in the last 24 hours? The Scrum of Scrums leader logs what tasks were completed and sends out an email to the company immediately following the Scrum of Scrums.
- What blocks were found in performing tasks in the last 24 hours. These are logged, reported, and followed-up after the meeting.
- What will teams work on in the next day. Team members volunteer for tasks. The ScrumMaster and the Lead Architect may help bring focus to appropriate tasks.

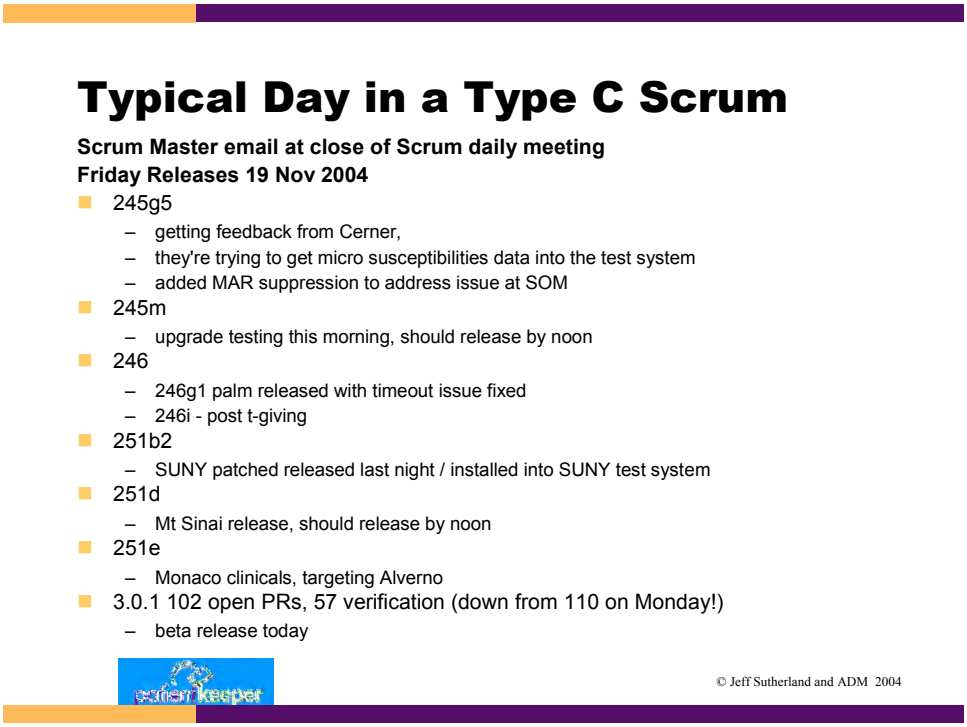


Figure 28: Summary of daily email after Scrum of Scrums meeting shows seven releases in progress simultaneously. All teams work on all releases and all releases result in customer deployment.

The Scrum of Scrums meeting takes place at the same time and place every day. An open space was secured by the development team for this purpose. Pair programming was done primarily on tasks with difficult design and coding requirements. Many of the developers stayed in the open meeting space for the entire day working together as a group. Innovative and open cube space and a few offices and conference rooms are provided for those who need quiet, focused time.

The rapid pace of delivery of production code releases initially created a Quality Assurance (QA) bottleneck. The solution was to assign an small QA team to every release. QA was expanded to four small teams of 2-4 people. This enabled them to work continuously on four of the top priority releases. In the Figure above, where six releases are being simultaneously developed, QA is doing final release testing and packaging on four of them. QA is part of the Scrum of Scrums and reports on daily status of ongoing releases.

4.5.2 Data Collection

A user group study and focus group analysis was performed for data collection for tasks, estimates, and updates that would be used to automate the standard Scrum burndown charts [17]. A wide variety of Scrum tracking tools had been used by members of the team in various companies over a 15 year period, none of them considered adequate. The 60 second requirement for data entry implied that a new application would not be possible, because simply starting up a new application might require 60 seconds.

The best application to use was one that developers had to use every day, the bug tracking system. In addition, the speed at which developers could do data entry was dependent on the questions they were asked, and the order in which they were asked. It was determined that only three questions would be asked as developers could answer them without thinking, they could give a gut level response:

- What is the initial estimate for this task if it is a new task?
- At this moment, how much time have you spent on this task?
- At this moment, what percent complete is this task?

These were the only additional data items to be collected daily from developers for tasks. All other data analysis and reporting was to be automated.

4.5.3 Tools for Data Collection and Reporting

PatientKeeper used the open source GNATS bug tracking system [100]. Since developers needed to use the bug tracking system daily, there was no additional time overhead for opening the application to enter task data.

GNU GNATS is a set of tools for tracking bugs reported by users to a central site. It allows problem report management and communication with users via various means. GNATS stores all the information about problem reports in its databases and provides tools for querying, editing, and maintenance of the databases.

Thanks to its architecture, GNATS is not bound to a single user interface – it can be used via command line, e-mail, Emacs, or a network daemon, but is usually used with a Web interface. Together with the fact that all GNATS databases and configuration can be stored in plain text files, it allows easy use and provides good flexibility. Basically, if the GNATS tools do not provide everything you need, you can add your own additional utilities using standard GNU tools. http://www.gnu.org/software/gnats/

A PERL expert on the development team was assigned to build utilities around GNATS to support Scrum. These were addition of required data items, new queries, minor changes to the user interface, and automated file dumps for management reporting via Excel. Sample data items maintained by GNATS are shown in Figure 3 below.

```
Message-Id:  message-id
Date:        date
From:        address
Reply-To:    address
To:          bug-address
Subject:     subject

>Number:        gnats-id
>Category:      category
>Synopsis:      synopsis
>Confidential:  yes or no
>Severity:      critical, serious, or non-critical
>Priority:      high, mediun or low
>Responsible:   responsible
>State:         open, analyzed, suspended, feedback, or closed
>Class:         sw-bug, doc-bug, change-request, support,
duplicate, or mistaken
>Submitter-Id:  submitter-id
>Arrival-Date:  date
>Originator:    name
>Organization:  organization
>Release:       release
>Environment:
   environment
>Description:
   description
>How-To-Repeat:
   how-to-repeat
>Fix:
   fix
>Audit-Trail:
appended-messages...
State-Changed-From-To: from-to
State-Changed-When: date
State-Changed-Why:
   reason
Responsible-Changed-From-To: from-to
Responsible-Changed-When: date
Responsible-Changed-Why:
   reason
>Unformatted:
   miscellaneous
```

Figure 29: Typical data items in GNATS for problem reporting by email or the web.

It was decided that sprint tasks would be treated like problem reports. This minimized new data entry requirements and allow tasks and bugs to be packaged together seamlessly for a release. Only three data items were added to GNATS for developer entry:

- Initial estimate
- Days invested
- % complete

The initial estimate was fixed at initial entry and could never be changed in order to allow for accurate historical reporting of estimates versus actual time to complete tasks. Two additional data items were added for reporting purposes. These are automatically calculated from the three items above.

- Days remaining
- Actual time to complete

If the initial estimate is 2 days, for example, and no work has been accomplished, the days remaining are 2 days. If a developer has invested 1 day and states that it is 25% complete, GNATS calculated the days remaining as 3 days. Initial estimates are automatically expanded based on real time data.

PatientKeeper gnats

User: gaspeslagh
Database: virtmed
Access: edit

Query Results

Can't find it? Create New Problem Report: create

12 matches found

PR	Category	State	X-Milestone	Resp.	Synopsis
10986 edit	v2_pkp	feedback	PKP-3.2	gaspeslagh	LK: can i have a sort order for the panels in the 'new panel' picker?
13073 edit	v2_pkp	open	PKP-3.3	gaspeslagh	FW: Enable patient sending via bluetooth
13102 edit	v2_pkp	open	PKP-3.2	gaspeslagh	FW: Uninstaller does not delete itself until after soft reset
10131 edit	v2_pkp	open	PKP-3.2	gaspeslagh	LK: Indicator flag not appear for sample data
10903 edit	v2_pkp	open	PKP-3.2	gaspeslagh	FW:fonts and tables
10925 edit	v2_pkp	open	PKP-3.2	gaspeslagh	LK:allow change field type
10949 edit	v2_pkp	open	PKP-3.2	gaspeslagh	LK: Edit filter categories
10960 edit	v2_pkp	open	PKP-3.2	gaspeslagh	LK: sort by ascending AND descending date
3929 edit	v2_pkp	feedback	PKP-3.1.3	gaspeslagh	LabKpr: Pending Status Category
9287 edit	v2_pkp	open	PKP-TBD	gaspeslagh	Add support for panel category editting.
9288 edit	v2_pkp	open	PKP-TBD	gaspeslagh	Add edit option for lab component types
10947 edit	v2_pkp	open	PKP-TBD	gaspeslagh	Q: Back buttons cause Treo 600 to reset

submit update
reset

Figure 30: Developer workstation task listing for assigned tasks for a Sprint. Right click on the mouse generates drop down list that allows any data item to be updated almost instantaneously.

The cumulative time remaining for a release can be obtained in real time by anyone in the company with access to GNATS. At PatientKeeper, that is every person in the company. The days remaining for all tasks assigned to a release are totaled to calculate cumulative backlog, the number charted on a Scrum burndown chart. Because there are thousands of tasks in the system and any tasked that is touched is updated every day it is touched, the phenomenon of statistical regression towards the mean [101] makes the summary data on cumulative time to release very accurate. It achieves the holy grail of accounting software, microcosting of every activity in a company [102].

4.5.4 Product Development Process Refinements

Product Management serves as the Product Owner at PatientKeeper and must provide functional specifications for features that sufficiently describe the user experience so that developers can begin design and coding. This typical means screens shots, workflow between screens, business logic, and data items required. A working demo is normally prototyped and approved by a physician user group before a feature can enter a sprint backlog.

The Product Owner controls the GNATS entries for a release and the bug flow into the system. Bugs initially go into a triage category and the Product Owner assigns them to a release based on priority, customer requests, implementation issues, and so forth. Features are initially placed into GNATS as placeholders assigned to a release. Developers can pick them up and translate them into tasks specific to a sprint.

Figure 7 shows a burndown chart generated by GNATS for PatientKeeper Release 6. It shows product backlog accumulating as placeholders until the Sprint started on 12 June 2002. At that time, developers began breaking down product features into estimated tasks for the next sprint. This drove the backlog up as new tasks are discovered that were unanticipated in original product feature estimates.

On 26 June 2002, the PatientKeeper CEO decided to commit the company to a major user interface rework during Release 6. When the Product Owner entered over 80 new tasks into the release the burndown chart shot up quickly in a two day period. This was visible immediately to the entire company because burndown charts were mailed out to the company on a daily basis.

This caused a lot of consternation because there was no way that this iteration would be a standard 30 day Sprint. PatientKeeper has a weekly MetaScrum meeting which includes leaders from all departments in the company where release priorities and features are reexamined on a weekly basis. It was determined that the value to the company of refining the user interface of the product was very high in a competitive environment and the sprint would be extended to 24 August based on the Scrum burndown chart. This would require the development team to have perfect development days from the beginning of July through the end of August.

An ideal developer day is the amount of work that can be accomplished if a developer works uninterruptedly for a normal work day. Most teams preplan to get 40-60% of a development day due to meetings and extraneous interruptions for team members. Getting 180 developer days with an 8 person team in 42 calendar days without overtime was not going to be easy. The policy at PatientKeeper was to sustainable development using a normal work week with night and weekend activity required only when rare emergencies occurred, i.e. production customers hard down in the field.

PatientKeeper had the advantage of smooth running Scrums with few interruptions other than release priorities. As a result, their normal velocity or number of days work accomplished per day per developer ran better than 60%. Years of historical data also showed they finished their tasks in an average of 85% of the original estimate. This often did not show up in calendar days due to release priority conflicts. However the leadership team realized that slippage was normally due to management prioritization problems, not developer slippage, and GNATS has mountains of data to prove it.

The solution was to focus the entire team on one release and over a 42 calendar day period, or 30 business days, the developers delivered 180 days of work for a velocity of 2 days of work for every 3 business days invested or 66% of the ideal. This was planned in advance, the MetaScrum leadership repositioned all customers at the beginning of July, and overtime was not required.

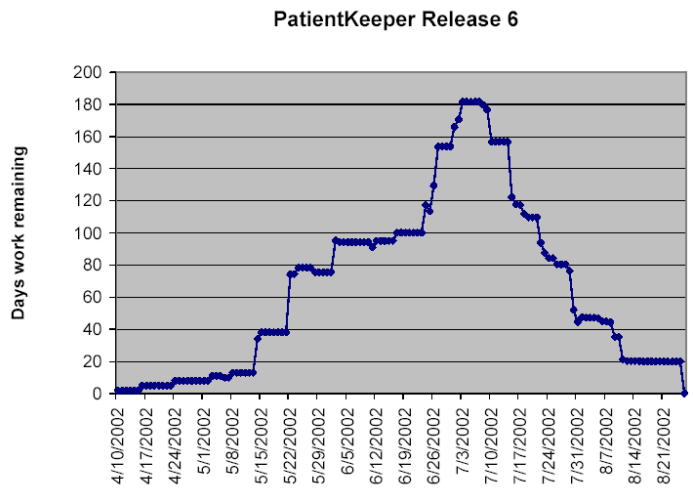


Figure 31: Scrum burndown chart autogenerated by GNATS. Product management was entering placeholders from product backlog until 6/12/2002 when the sprint started. It extended beyond 30 days for reasons described elsewhere. The pain of this Scrum reconvinced everyone that 30 days is the maximum Scrum length.

The capability of total transparency where all data is available to everyone extends the concept of global visibility during a Scrum to the entire company. This allows all parts of the company to replan activities routinely. In the case of PatientKeeper, new Sprints can be started, changed, or destroyed in the weekly MetaScrum without disrupting the focus of the Scrum teams.

4.5.5 Project Reporting

The GNATS reporting system was refined to allow sophisticated management reporting. Two of the most useful examples are (1) tracking the quality of a product portfolio and (2) automated generation of Gantt charts for MetaScrum planning.

4.5.5.1 Tracking Quality of a Product Portfolio

A useful measure of product quality, code stability, and forward progress is a chart that shows arrival of new development tasks, completion of development tasks that change status to verification (where they become the responsibility of the QA team), and closing of tasks when testing by the QA team is complete. The cumulative number of outstanding defects has been divided by 10 in Figure 8 to allow charting of the cumulative total in the same range and daily defect arrival.

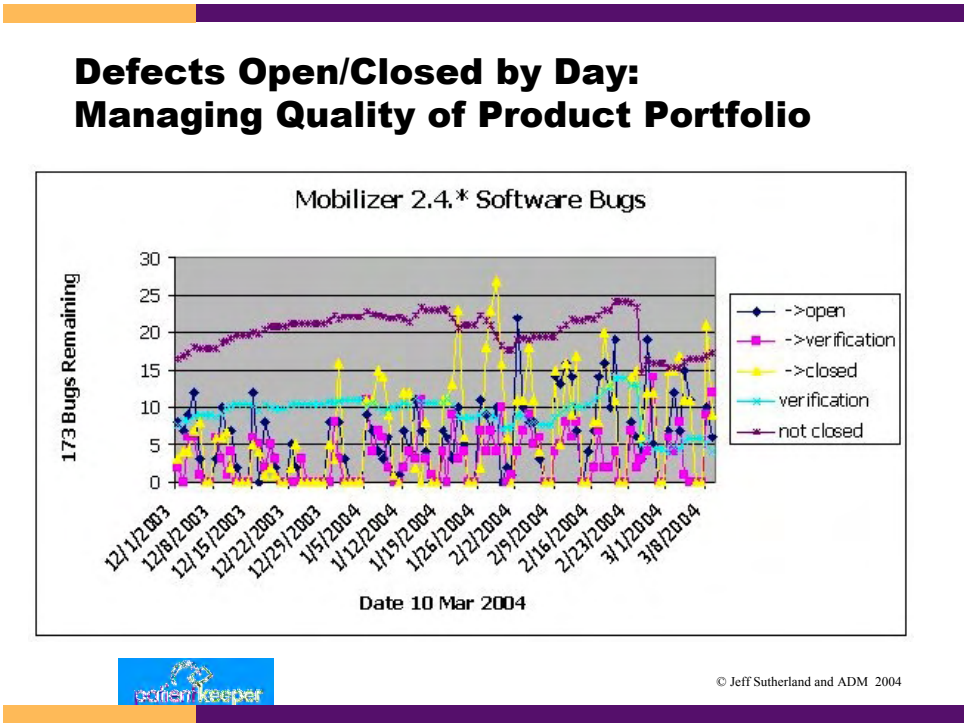


Figure 32: Daily arrival of defects along with cumulative defect count. Company objective is to drive total bug count across a deployed portfolio of releases below 100. This is challenging as there are about 100 hospitals deployed on the 2.4.* releases.

4.5.5.2 Gantt Chart for MetaScrum Planning

Gantt charts are useful for planning purposes. However, they are poor for tracking software projects because dependencies change on a daily basis. A full time developer can be absorbed keeping Microsoft Project up to date for a single Scrum team and Scrum was designed to eliminate this wasteful activity. For the last six years, PatientKeeper has evaluated whether to have any project management other than Product Owners and ScrumMasters. The decision has always been that they are unnecessary waste.

A MetaScrum can find a Gantt chart useful, but only if it is machine generated. A human generated Gantt chart is inaccurate in the beginning and completely outdated within a week in a fast-paced company. An effective Gantt chart can be calculated in real time based on data capture in the reporting system.

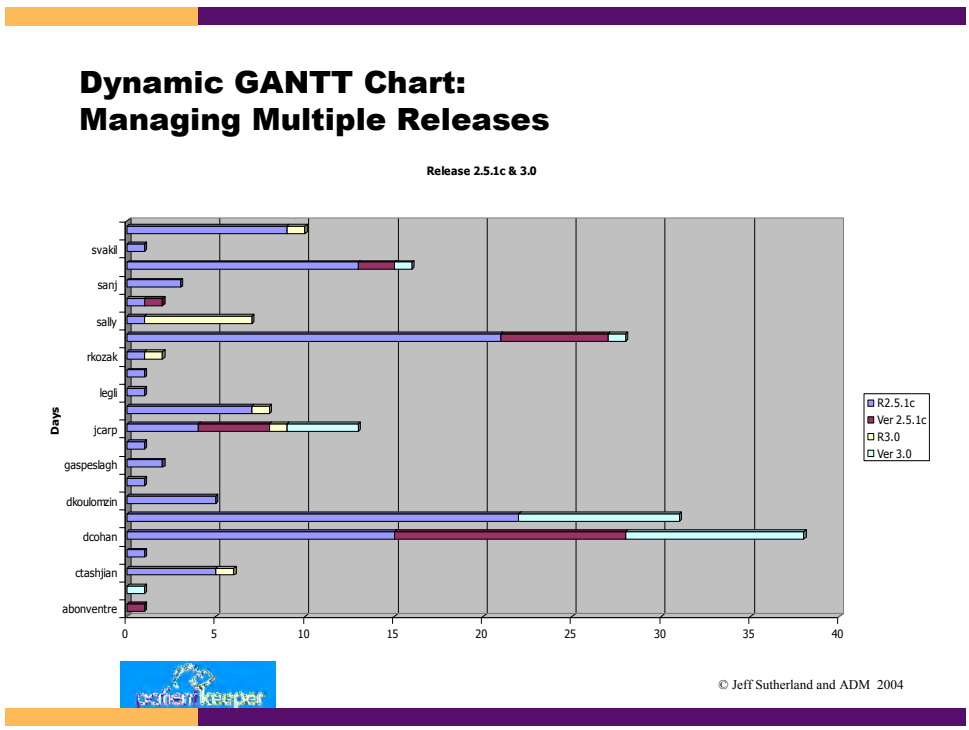


Figure 33: Dynamically generated Gantt chart. End points reflect anticipated release date in days from day of generation.

Dynamic Gantt charts can be generated by staff members for a small team, or by teams for a company. The chart above shows two releases broken into development tasks and QA tasks. The X axis is days from generation day. The Y axis is staff members by name showing assignments for the team.

The result of an automatically generated Gantt chart is a surprise to traditional managers. It will show most individuals are loaded only 1-3 days out. They will choose their next task when they complete their current task. Key people like the lead architect or QA manager will have tasks queued up to be passed off to whoever is ready to take them "just in time."

When PatientKeeper managers were asked if they wanted to manage resources to allow an autogenerated Gantt charge to show release dates they were counting on, they immediately declined, noting that disruption of a self-organizing system would radically cut the velocity of the team and create unnecessary work for managers. They gave up the notion of trying to use a Gantt chart to plan releases and went back to the Product Owner's roadmap for release planning. This is a milestone based timeline that shows the Product Owner's best estimate of release dates with specified highest values features based on known team velocities and project dependencies.

4.6 Type C Scrum Rationale

As noted in our Pattern Languages of Program Design paper [25], "it is very easy to over- or under-estimate, which leads either to idle developer time or to delays in the completion of an

assignment. Therefore, it is better to frequently *sample* the status of small assignments. Processes with a high degree of unpredictability cannot use traditional project planning techniques such as Gantt or PERT charts *only*, because the rate of change of what is being analyzed, accomplished, or created is too high. Instead, constant reprioritization of tasks offers an adaptive mechanism that provides sampling of systemic knowledge over short periods of time. Scrum meetings help also in the creation of an *anticipating* culture [103] because they encourage *productive values*:

- They increase the overall sense of urgency.
- They promote the sharing of knowledge.
- They encourage dense communications.
- They facilitate honesty among developers since everyone has to give a daily status.

In a Type C Scrum, the urgency, sharing, communications, and honesty behaviors are extended company wide. "From the Complexity Theory perspective [104, 105], Scrum allows flocking by forcing a faster agent interaction, therefore accelerating the process of self-organization because it shifts resources opportunistically through the daily Scrum meetings.[25]" When extending company wide, the entire company can self-organize on a weekly basis. The following behaviors become commonplace:

- There is never an unexpected late release as problems are seen long before the release date. The company self-organizes around the issues raised in the MetaScrum.
- Changes in customer requirements are reflected immediately in product backlog and relevant Sprint backlog. Decisions are made to reorganize on a weekly basis in the MetaScrum.
- Company imperatives and management changes that affect product backlog are made only in the MetaScrum. This eliminates most politics, lobbying, and closed door meetings.
- Customer impact and schedule impacts are deal with immediately in the MetaScrum at the time of decision. The CEO, sales staff, and account management walk out of the meeting with assigned tasks to deal with customers affected by decisions.

4.7 Type C Scrum Resulting Context

The move to a Type C Scrum to improve development productivity had far reaching effects on the company making it more flexible, more decisive, more adaptable, and a better place to work. The same effects commonly seen on Scrum teams were reflected throughout the company. Project management was totally automated. The result is paperless project management and reporting, largely without human intervention. Scrum execution has become exceptionally efficient and the automated tracking system has become mission critical. Burndown charts have evolved to frame the entire status of a project on one chart. The chart below instantaneously reflects project state for Release 3.20 at a glance to those familiar with the data. With all tasks

entered at 16 hours or less and bug fixes typically less than a day, the aggregate number of tasks can be monitored and downward velocity is highly predictive of delivery date. Information is presented as follows:

- **Dark Blue Diamond** – Release 3.20 current open – cumulative work remaining
- **Yellow Triangle** – Release 3.20 daily closed - items closed by QA each day
- **Purple Star** – Release 3.20 total closed - cumulative closed (on scale at right)
- **Pink Square** – Release 3.20 current verification - current total in verification (items QA needs to test and close)
- **Light Blue X** – Release 3.20 daily open – new tasks opened per day

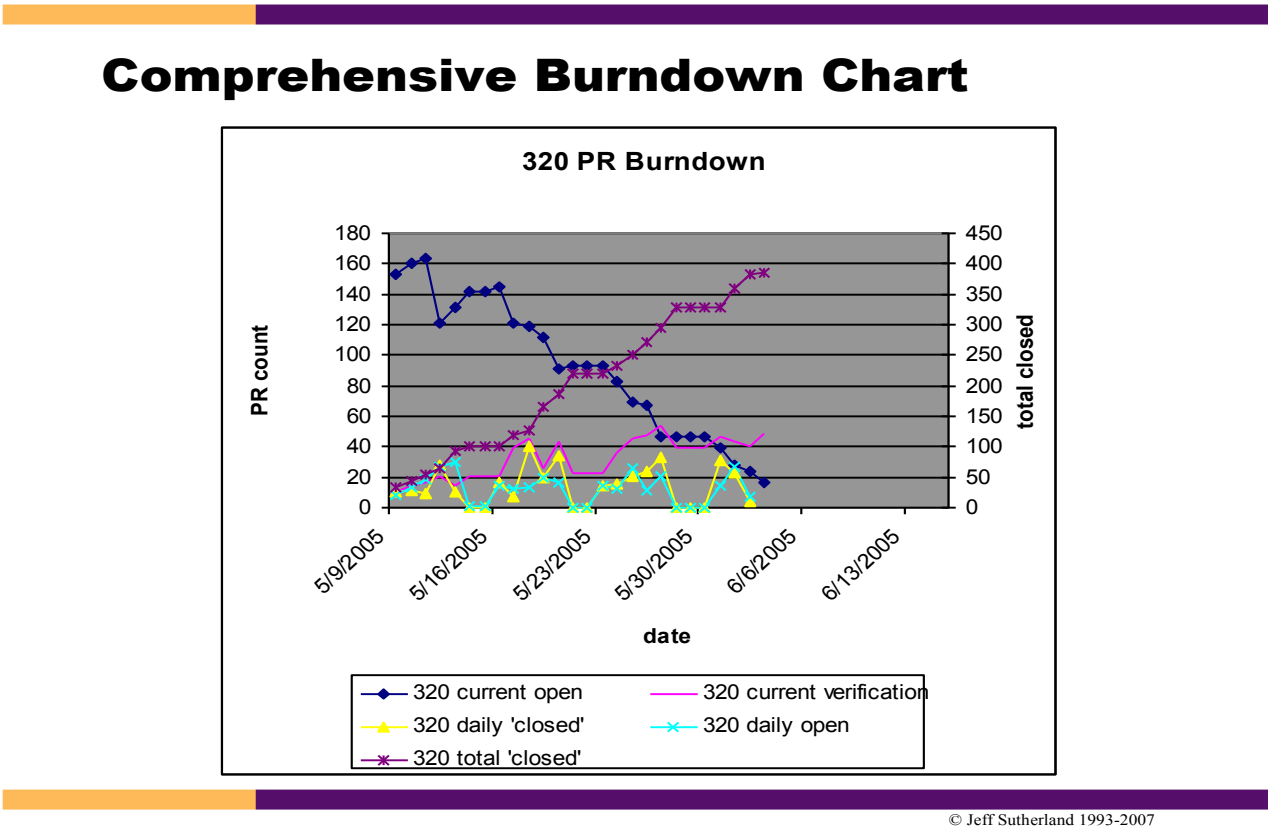


Figure 34: Comprehensive Burndown Chart showing daily task inflow/outflow and cumulative project churn [15].

The cumulative closed (right scale) is much higher than the starting number of about 150 tasks (left scale). The reason for this is that the Sprint Backlog minor changes are constantly coming into the Sprint Backlog for the following reasons:

- QA is finding bugs, often generating multiple tasks that can be closed with one developer fix.
- Product development is adding tasks primarily because of customers moving in and out of the mix for go-live at end of Sprint (this is not allowed in Type A and B Sprints).

- Development is discovering new tasks as they flesh out technical design.

The cumulative closed tasks is an indicator of the churn on a project and the reason why Brooks [5] notes that development always take three times as long as initial estimates. Automated reporting and rapid turnaround can radically reduce time to complete new tasks. Note the strong downward velocity on the Burndown Chart despite project churn. PatientKeeper was able to move quickly into the marketplace and achieve leadership in the healthcare mobile/wireless market [11] through delivering over 45 production releases of the PatientKeeper Platform in 2005 for large enterprises such as Partners Healthcare in Boston, Johns Hopkins in Baltimore, and Duke University Health System in Durham. Gartner Group put PatientKeeper as the leader in their "magic quadrant" for the industry segment. Type C Scrum was a key contributor to this success.

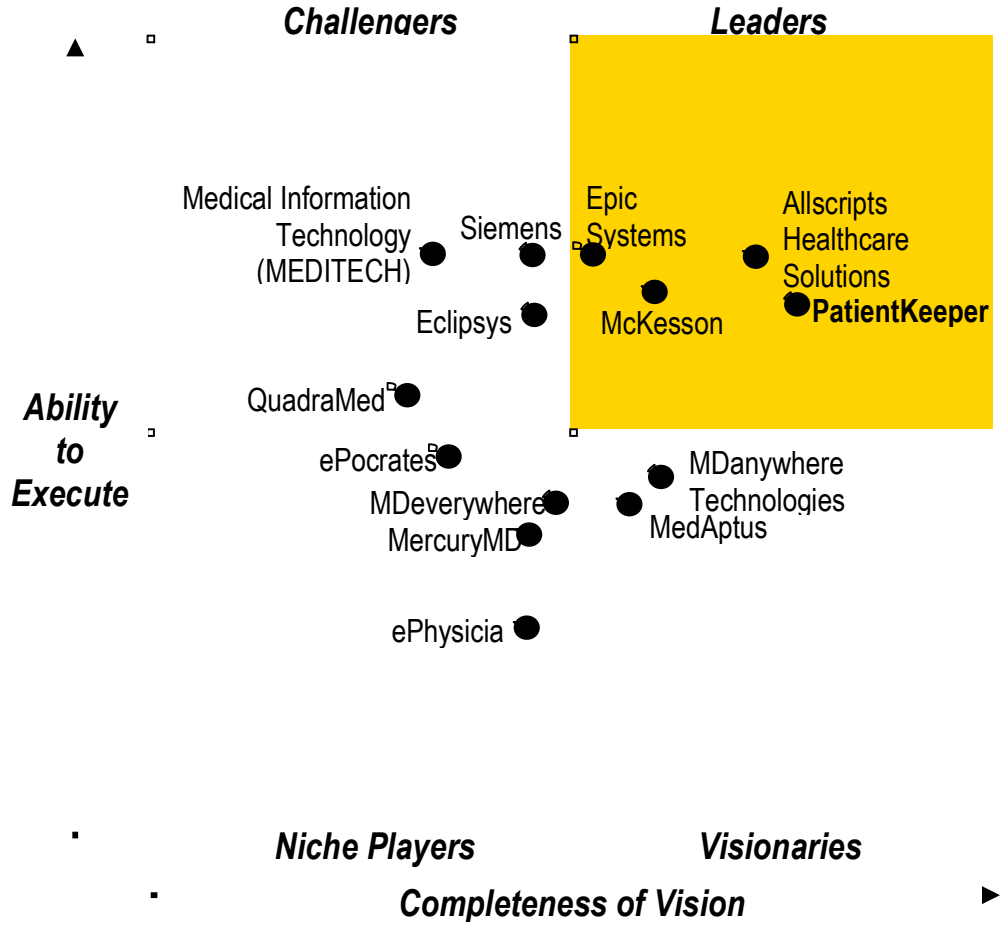


Figure 35: Gartner Group "magic quadrant" for healthcare mobile applications [106].

5. Conclusions

Moving to a Type C Scrum is not for the faint of heart. It requires Scrum teams that can execute a standard sprint flawlessly, an automated data collection and reporting system that is easy to implement and update, and a corporate culture that embraces change. Going to a Type C Scrum

will transform a company in an organization where Scrum becomes mission critical for the entire organization, not just software development.

Gregor Rothfuss provided an excellent summary when seeing the reporting mechanisms for a Type C Scrum for the first time [89]:

i was a guest at the Agile roundtable near Boston last night. The event drew a crowd of veteran software engineers, i was the youngest in attendance by about 20 years.
ken schwaber outlined his and jeff sutherland's Scrum approach, which struck me as interesting and worthwhile to follow up on.

jeff sutherland, CTO of patientkeeper, demonstrated how he manages his teams of developers with GNATS. jeff figured that developers loathe red tape, and had the goal to limit the effort required to 1 minute per day for developers, and 10 minutes per day for project managers.

and he was not using gantt charts to achieve this either. calling gantt charts totally useless for project management beyond giving warm fuzzies to the client, he explained how he leveraged their bug tracker to double as a means to keep track of effort.
each morning, developers review their tasks and update the work remaining estimates which have a granularity of one day. the project managers, in turn, analyze the reports that GNATS automatically creates. reports such as number of new tasks vs. closed tasks, total work remaining and other metrics that can be derived from the task data.

tasks are the cornerstone here. jeff was able to demonstrate to the business side that the high level business goals were off by 100% with their effort estimates, while the low-level tasks achieved an accuracy of 10% on average. this led to enthusiasm from all parties to drill down on any project and get to the task level ASAP to get meaningful estimates. and, like psychohistory, project management is inherently stochastic.

'nowhere to run, nowhere to hide'
the level of transparency of this system is unprecedented. with everyone in the company able to see on a daily basis how much work was remaining and what the roadblocks were, the initial fears that developers would be pounded on by management turned out to be unfounded. instead, the transparency enables everyone to do real-time adjustments and to detect problems early, which has taken a lot of politics and second-guessing out of the equation.

when analyzing a project, jeff focuses on burn down, the part of a release where open tasks are relentlessly driven down to 0 by a joint effort of developers and business people. the corresponding graphic (roughly a bell curve) illustrates the importance of the burn down nicely, adding weight to jeff's assertion that burn down is the only thing that matters to get a release done in time.

"which prompted me to ask for advice on how to drive an open source release as a release manager. people are not exactly required to do your bidding, but metrics may help there too. collect these useful data points, as the bugzilla-bitkeeper integration is doing, and let them speak

for themselves. peer pressure and pride in workmanship will take over from there. that's the idea anyway…"

Key features mentioned in the Rothfuss report are:

- *Unprecedented transparency*
- *Companywide visibility*
- *Metrics driven decision making*
- *Peer pressure and pride in workmanship driving productivity*

Type C Scrum increases speed of development, aligns individual and corporate objectives, creates a culture driven by performance, supports shareholder value creation, achieves stable and consistent communication of performance at all levels, and enhances individual development and quality of life.

Chapter 7: Case Studies

One of the most interesting things about Scrum is the unique case studies that have been published at IEEE conferences. Scrum is used by some of the most productive, high maturity, and most profitable software development teams in the world. It powers:

- The most productive large development project (over a million lines of code) ever documented.
- The most unique CMMI Level 5 implementation on the planet.
- The most profitable software development project in the history of software development.

The SirsiDynix project reimplemented a large system that supported over 12,500 libraries in 42 countries with 120 million users. In the middle of the project, the development team was doubled in size with teams from StarSoft Development Labs in St. Petersburg, Russia. Their velocity more than doubled the day the Russian teams came online. The lines of code delivered per developer was as high as the best small team collocated Scrum projects.

Systematic Software Engineering in Aarhus, Denmark, spent seven years and over 100,000 person hours of process engineers to achieve CMMI Level 5 certification, reduce rework by 80%, and improve productivity by 31%. Within six months after a Scrum Certification course they had reduced planning time by 80%, defects by 40%, total cost of a project by 50% while simultaneously enhancing customer and employee satisfaction. They now bid Scrum projects at 50% of the cost of waterfall projects.

One of the most interesting Scrum implementations is Google's AdWords implementations. This application drives the majority of Google revenue growth and helps create market capitalization that is higher than Intel and just below that of Chevron, the most profitable oil company in the world. The AdWords project, powered by Scrum, has distributed teams in five locations and interfaces with virtually all Google products on every release. As a result, the Google project manager needed to insert more structure than is usually associated with Google teams. His seamless introduction of Scrum based on resolving the highest priority impediments observed by the teams resulted in an implementation that no longer needed a ScrumMaster to function. The teams ran by themselves.

Ssh! We are adding a process… (at Google)

Mark Striebeck, Google Inc.
mark.striebeck@gmail.com

Abstract

Google is very successful in maintaining its startup culture which is very open and engineering-centric. Project teams don't have a project manager, but organize themselves and communicate directly with all stakeholders. Most feature decisions are made by the engineering teams themselves. As well as this works for products like search, gmail … it creates issues for the AdWords frontend (AWFE) application. AWFE is much more product management and release date driven then other Google applications. This presentation discusses how we carefully introduced agile practices to coordinate the AWFE development teams and made the process more efficient and predictable.

1. Introduction

Google is known for its startup culture and its efforts to maintain it. In terms of the Agile Manifesto Google is almost entirely on the "left hand side" (with the exception of "Working Software"). Traditionally, project teams do not have a project manager, but organize themselves and communicate directly with all stakeholders. Even now, where Google has more then 6000 employees in numerous offices around the world, Google is still very engineering driven. Many new product ideas come from the 20% projects[4] of its employees.

The overall mindset at Google is to have as little as possible standard processes as possible. The reason is that the individual engineering teams will know best what is right for them. Upper management on the other side has trust in its engineers that they would not abuse this autonomy but do what is best for their project and the company.

AdWords is different. Being a B2B application, means that it needs much more business focus, sales material has to be updated, support has to be trained, and external communication about major features has to be prepared (forums, blogs and emails).
Therefore AdWords had a few standards:

- From the initial product idea, the product manager together with a UI designer and usability specialists creates almost final UI mockups. These mockups are used for a final project review by senior management and then given to engineering for implementation.
- During the whole project lifecycle, the product manager holds weekly meetings with all stakeholders (engineering, QA, UI, support, marketing). These core team meetings are the main communication channel. All major product decisions are made or at least

[4] Every Google employee is encouraged to spend 20% of his/her time on a personal project. This project should not be too closely related to the employees' actual work.

discussed here. Lots of change requests come from these core team meetings during the project lifetime.

- Although, the core team sets initial release dates (with input from engineering), the final release date is determined by engineering progress and quality of the code. Given the scale of AdWords (number of users, business relevance, load, infrastructure); a small bug can have very severe consequences. Therefore features are rather delayed then released with insufficient or even unknown quality.
- This level of process worked well in the beginnings of AdWords. But the AdWords product development outgrew this ultra lightweight process
- The application code consists of more then 500KLOC

The engineering team is distributed in 5 offices worldwide – there are constantly 15-20 major projects ongoing plus maintenance and small improvements.

And the AdWords application and development team is still growing…

The unpredictability of launch dates caused more and more concern. Nobody wanted to lower the high quality standards. But the initial release dates needed to be more reliable and delays should at least be known and communicated much earlier.

Because of its size and complexity AdWords has fairly large management team (for Google standards). In order to be effective the management team needed much more visibility into the projects and their status.

Finally, the rate of change is very high in AdWords. Teams who work on a project for a few months might find that they have a lot of cleanup to do before they can finally launch. Not so much because of code integration issues (the AdWords team runs a fairly comprehensive continuous integration suite) but because of feature changes. Often, projects that run for a long time have to play catch-up with all the feature changes before they release. In a few cases this lead to significant delays.

2. First agile attempts

Trying to introduce a process in a start-up environment such as Google often meets resistance. Because of the googley way of developing software, many engineers simply do not believe that any formal process can have a benefit but will only slow them down.

When I took on my first projects at Google I was just a few months with the company. The engineers did not know me at all. But it was interesting to see how the Google culture helped me here: A big part of Google culture is trust. This goes through the whole organization. And although I was new to Google and AdWords, the engineers and PMs trusted me that I would do the right things. Or better: they trusted the people who hired me that I am someone who would do a good job.

So, my strategy was to get as little involved as possible in the actual coding part and to start with a few practices that would just help us to track progress and show issues. Then we would

introduce individual agile practices to fix such issues during development. I decided to start with the following practices:

A release backlog and burndown charts [i]. These two tools provide high visibility into the development progress for the project team, but also outsiders. Using simple wiki pages to store the backlogs allowed the engineers to update their progress in very little time. I decided to measure the burndown rate by tasks complete, not feature complete. Measuring progress in feature complete has many advantages but also forces a team to change their development process a lot. It was one of the areas where I decided to rather introduce this practice later in order not to overwhelm the team.

In past projects I made very good experience with estimating features/tasks in points [ii]. Especially in an environment like AdWords, where engineers are often interrupted by meetings or tech talks, real time estimates are a problem. If the burndown graph tells us that we are implementing 3 days of work per week then it often leads to discussions what the team is doing the other 2 days. Or people try to match their updates to real days. Points are a good abstraction layer that avoids any such discussion.

Scope changes are included in a controlled way by first estimating them, adding them to the backlog. Here, the burndown charts helped tremendously to get a quick assessment of the impact.

A weekly development checkpoint meeting to plan the next week and work on scope changes. These checkpoint meetings were attended by the engineers, QA, PM and UI. At this point I did not introduce real iterations. My personal experience was that changing to iteration-based development is a significant change for developers and QA. It sounded too heavy to introduce at this point.

For the adoption of these practices, I tried very hard not to implement anything top-down but to get buy-in from engineers and the product managers. The initial changes sounded reasonable to the engineers. Because I was managing several projects, I could not be too closely involved in the development activities itself. This probably worked to my advantage – the engineers realized quickly that I would not try to tell them how to do their job, but that I only structure the project in a certain way which was not too intrusive. Also, one of the goals was to keep the self-organizing character of teams intact. After all, this is a big part of Google culture and our agile adoption approach would have failed if we had severely impacted it – no matter how successful the projects would have been.

This approach also helped me to work with several projects at the same time. Many meetings regarding UI, features, design… took place without me. Only when we discussed scope, scheduling or planned the next steps, I was there and was usually leading the meeting.

2.1. The guinea pig projects

Changes at Google are often done in some kind of guerilla approach: one project team adopts something new. If it works, other project teams get interested and will try it as well. Therefore, we started only with two projects:

Project A: This was a very new piece of functionality which did not overlap with existing features. The UI was fairly complex; the engineering team consisted of new recent college graduates working in a remote office.

Project B: This project was a simplified version of AdWords. It was heavily integrated into existing features (we basically had to think about every other feature and had to integrate or disable it). The team consisted of experienced engineers. Some of which had already work for some time at Google, others were new to Google).

2.2. The first process steps

In both projects, we used the UI mockups to generate the release backlog by dissecting the screens into individual features. This pre-development process is very well established at Google and it seemed too complicated to make this part more agile.

The release backlogs were stored in wiki pages which made it very easy for engineers to update them. From these wiki pages we automatically generated burndown graphs to visualize the project progress. The concept of giving status updates in work left and not in work completed was initially strange to both teams. But the engineers quickly realized the advantage.

As stated earlier I did not introduce iterations at this time. Instead I installed weekly checkpoints with the development team (PM, UI, Engineering and QA). In these checkpoint meetings, we discussed progress, additional feature requests and other issues. Additional features were estimated and added to the release backlog. I extended the burndown graphs and used a variable floor to indicate the scope changes. The graphs gave us quick feedback what the estimated impact of these additional features was.

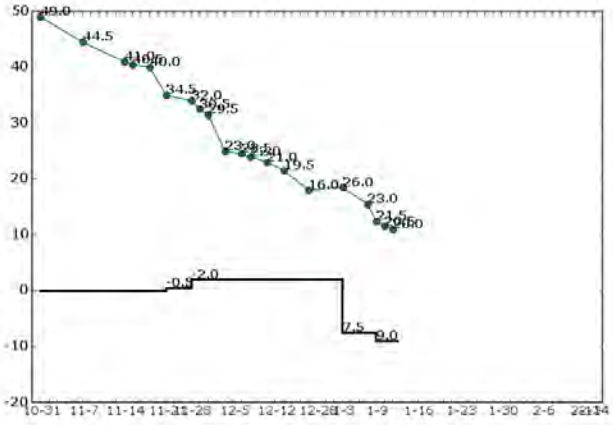


Table 1: Burndown graph with variable floor

Although I did not try to implement an immediate testing of implemented features, I wanted to get away from the purely phased approach where testing starts after development is finished. To

push for this, we setup staging servers that were rebuilt on a nightly base with the latest code. These staging servers were used for testing the application but also for UI walkthroughs[5].

Usually, they are performed towards the end of a project when the system is nearly complete. But because we staged the application early on and implemented end-user features (from the UI mockups) we could start with these UI walkthroughs much earlier and gather important feedback for the further development.

2.3. Issues to overcome

In both projects we faced similar issues:

Customer / Product Owner concept
Most agile processes have this role which is responsible for features, prioritization and ultimately scope vs. release date decisions. It is usually an individual or team outside of the development team. But at Google, many of these responsibilities rest with the team leads. The product managers usually have more then 10 projects at the same time. This does not give them the bandwidth that the product owner role requires. Also, they trust the tech leads and UI designers enough that they will make good decisions (often, when I asked a product manager for prioritization of a feature, he turned to his tech lead and simply asked "what do you want to do?").
This gives the planning and prioritization meetings a different dynamic. Often, the tech leads do not see the need to make such decisions during the planning meetings as they know that they will be involved enough during development itself that they can make such decisions at a later point. I usually drove the team to make at least those decisions which are necessary to create good effort estimates and priorities for the backlog. I always wanted to leave the weekly checkpoint meetings with good updates to the release backlog.

Retrospectives
For me, frequent retrospectives [iii] became such an important part of software development that I tried to install them in the weekly checkpoints from the beginning. It would have helped a lot with improving our process through constant feedback.
But both teams were not (yet) used to having a formal development process. The weekly retrospectives usually turned into a status report from the last week but very little about the process itself. This was aggravated by the engineering centric culture at Google. When an issue comes up, most engineers at Google only consider technology to fix it.

After a few weeks, I silently dropped retrospectives from the weekly checkpoints. I decided to wait until the teams embraced the concept of a development process and that they own it and could change it to fix problems.

Constant scope increase
In both projects, the scope increased significantly (more then 30%) during development. Interestingly, these scope changes were not the result of additional feature requests by the

[5] UI walkthroughs are live demonstrations of the system with the whole core team to gather feedback and uncover usability issues early enough.

product managers. Most additional tasks were the results of oversights during the release planning:

The engineering team missed features in the UI mockups when we created the release backlog Integrations into other AdWords features were overlooked. Also, the rate of change in AdWords is very high. During development others areas of the application changed and we had to change our integration as well.

Most of these additional tasks could not be down prioritized for a later release but had to be added to the release.

In both projects, this lead to several postponements of the release date as no other feature could be dropped from the first release.

Although, there was considerable frustration about these delays, both project teams and management appreciated that we at least knew about these postponements early enough and not just the week before release. The burndown graphs gave a good visualization and the release backlogs made it easy for everyone to understand what was left to be implemented.

The backlog was handy as things came up over time and as we dived deeper. One function was to not loose the line items but more important it was useful for the team to see how many un anticipated issues cropped up and have a good snap shot in time.

Product Manager

2.4. Working with the remote team

As stated earlier, project A was implemented in a remote location. The rest of the core team was in our headquarters. Initially, I was concerned how that team would react to my leadership – if they would appreciate it as much as the other team or if the would regard it as a heavy-handed approach from headquarters.

To my surprise I did not encounter many issues with this project. Only providing tools to get more visibility into development progress and facilitating planning meetings seemed to be the right level to give the remote team enough room to work mostly autonomously. Also, I could make myself very useful in facilitating lots of communication with other engineers in our headquarters. The team realized quickly that I indeed tried to help the project progress and not to control them remotely.

3. Adding agility – one practice at a time

3.1. Daily standup meetings

Both project teams initially rejected the idea of daily standup meetings [iv]. They were seen as an unnecessary overhead.

But during development we discovered issues in the weekly checkpoints from the past weeks: QA tested unfinished features or was not sure how to test new features

Engineers who worked on related features worked on the same refactors. The AdWords engineering team has a very healthy culture of constantly refactoring the code. The downside is that two engineers who work on related features often start to improve the same code.

Engineers could not continue with their implementation because they depended on a task from another engineer. Often enough, the other engineer was not aware of this dependency.

It was clear to everybody that these issues could have been avoided had the team communicated earlier. At this point it became easy to convince both teams to try out daily standup meetings and to include QA in these meetings.

The first standup meetings were quite lengthy. Everybody had a lot to talk about and had problems to focus just on a quick status update ("done", "to-do", and "issues"). But after a few days nobody had a big baggage anymore and everybody realized that there is not much to talk if you restrict yourself to the past 12 hours and next 12 hours. Several issues were resolved or at least uncovered during these meetings. After a couple of weeks, both projects did not need a reminder anymore but made the standup meeting part of their daily routine.

3.2. Small steps – Completely finishing a feature/task

In project A, the progress looked very good. Initially, we estimated 3 weeks for a set of screens. When we did low-level estimates, we came to 40 points. After the first week, the team did 8 points – in the second week 7.5 points. I looked as if the initial estimate was too low and the team would need 5 instead of 3 weeks.

Interestingly, the tech lead of the team was convinced that the screens could still be implemented in 3 weeks (i.e. all remaining 24.5 points in 1 week!) quote: "It just does not feel that much anymore".

After week 3, the team was not done. The team implemented another 9 points. The velocity looked very stable: ~8 points per week.

To my big surprise, the tech lead announced in the core team meeting once again that his team will be done in one week…

It took me some time to learn to trust the burndown graph and to question my gut feeling when a feature would be finished.

Tech Lead

The fourth and fifth week showed a significant drop in velocity: 4 points and 2.5 points! It turned out that the team did not completely finish the tasks: tests were not written, code was not

reviewed (which is mandatory at Google), features were not completely integrated. This caused the burndown graph to go down because we did not measure progress in finished features, but in tasks.

This caused a further delay and the screens were finally implemented after 7 weeks. This additional delay caused some concern with the core team. To avoid this situation I added a green/yellow/red color coding to the burndown charts to indicate how many tasks are new/started/finished. This made it very clear if velocity was high because many features are partially finished or if the team completely finished a feature before moving to the next one.

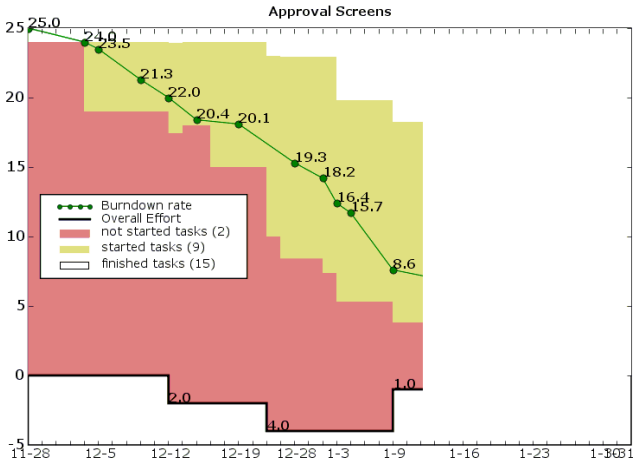


Figure 2: Indicating started and finished tasks

The team responded very positively. It was quite a shock for the engineers to see that up to 80% of all tasks were in a 'started' state. They started to keep the corridor of started tasks as small as possible.

Overall, this was a very healthy learning experience for the team. It showed them the difficulty that we tend to have when trying to estimate a release date instead of deriving the release date from effort estimates and progress. It also showed them that we can only measure progress well, if we completely finish tasks and not leave small bits and pieces around which sometimes turn out to be much larger then we thought.

3.3. Spikes

In the weekly checkpoint meetings we often discovered that tasks took much longer then initially estimated. Or the team had problems with estimating a new feature.

Initially, the engineers just wanted to pad estimates for such unknown tasks. Often enough, these padded estimates were much too high or still too low. And everybody could see that they lowered the usability of our burndown graphs significantly. So, we added in a spike (an investigative task) to help determine what the effort for the implementation would be. Especially

when the scope continued to grow, everybody realized the value of getting a better estimate of implementing a feature before actually starting to work on it.

4. Release experience

The two projects had somewhat different releases:

Project A)
The team had fixed many bugs already during development, only few bugs were discovered in the final integration test phase. It was a very smooth launch.

Project B)
Because of the integration into all other AdWords features, QA found many issues during development – most of them through exploratory testing [v] (i.e. not really tied to a particular product feature). The team tried to keep the bug backlog under control but did not want to fix all bugs. When we came close to launch, we had to review the bug backlog several times and down prioritize many bugs. Until a few days before launch it was not clear if we could fix enough bugs to release it.

At least the team did not encounter any issues that required a complete redesign of some area – which could have easily happened for such a far reaching feature.

Still, the overall release experience was very positive. Both projects were very successful in production and had very few issues.

5. Feedback and next steps

I held post-mortem meetings with both projects. In these meetings I focused the teams on listing positives and negatives and not jumping to discuss solutions immediately. From the overall list, the teams selected the worst issues and best practices to keep:

Positive
Project Management and tools (burndown charts and backlogs)
Early QA and availability of a staging server
Teamwork and collaboration

Negative
Unclear or non existent prioritization
Felt as if team missed release date several times
Too risky at end because of bug backlog (Project B)

It was very encouraging that both teams found the overhead of maintaining and updating the release backlogs worth doing.

Burndown charts made it easy to see when were making progress, and gave us a nice sense of satisfaction and completeness.

Engineer

And, furthermore that the process did not impact the great teamwork and collaboration that Google teams have. Also, the effort of maintaining a dedicated staging server was appreciated. The engineers from both teams were very positive about the early testing and feedback by QA that the staging server afforded.

I think it took some time getting used to the approach of testing so early in development, and also making sure that QA and dev were on the same page. I think that our daily standups and also having QA co-located with dev has helped greatly here.

Engineer

6. The second version

From the feedback of the post-mortem meeting I tried to modify the development process further to address the worst issues.

In both teams I gave at this point a presentation about a full Scrum process [vi]. During the first projects there were many tech talks at Google about agile development (by internal and external speakers). Both teams got very interested in it. They could see that their practices fit into agile development but heard a lot about other practices too. Also, the very positive feedback of my project management style and tools showed me that the engineers trusted me and my guidance. In both teams we discussed which additional practices to adopt:

Product/Release Backlog
To address the prioritization issue, I worked with the product managers of both projects to organize their requirements in prioritized lists. It took a little bit of time for them to get used to it, but was not a major effort. The core team members liked the backlogs a lot. It gave them much more visibility and input into development. Initially, there was still the desire to make each feature high priority. But soon everybody realized that even if a feature is not included in the current iteration, it will still get done fairly soon.

Iteration based development
This was the hardest practice to introduce. Without practical experience it is hard to explain why iterations are better than scheduling the whole release at once and adding to it when necessary. But with the feedback about missing deadlines and too many bugs, I could explain how an iteration based approach would address these. The concept of not only implementing but also testing and completely fixing features within the same iteration sounded very appealing to the engineers. Although, they were somewhat skeptical of this high-quality approach, both teams wanted to give it a try.

The teams soon realized the advantages. The planning meetings became much more focused than the weekly checkpoint meetings from the previous projects. No time was wasted with discussing the same feature for 5 weeks but never implementing it. Or to discuss and design a feature that finally gets dropped.

We agreed to start with 2 week iterations. This synchronizes well with the 2 week release cycle of AdWords. We are finishing the iterations with the code freeze for the next push. This means that a high-priority feature that gets put on the product backlog can be implemented and release within 4 weeks without any interruption.

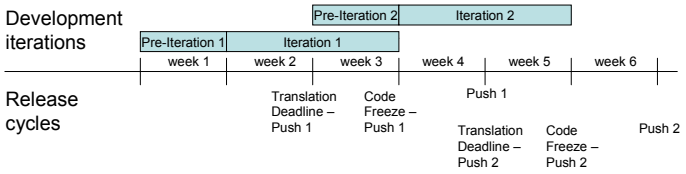


Figure 3: Synchronized development iterations and release cycles

Retrospectives
After the previous projects, both teams had some experience with a defined development process and that they can influence/change it. I started the iteration planning meetings again with a retrospective and this time it was much more fruitful. Most contributions were about how we develop our application and how we can improve that.

Review of iteration features with core team
In the first projects, we reviewed the application by clicking through it during the core team meeting and collected some feedback. Now, with the iteration based development we do these reviews at the end of each iteration and only on newly implemented features. This made the reviews more focused and gives us feedback early enough so that we can integrate it in the next iteration.

Testing tasks for features in same iteration
In order to test features in the same iteration as they are developed in, we added testing tasks to the iteration backlog. The QA engineers were asked to provide effort estimates for these tasks so that they can be included in the burndown chart.

Overall, the teams could see how these process changes would address the negative feedback from the post-mortem meetings. Both teams did not fully understand how these practices would work together but agreed to give it a try.

At this point I took on a third project where I implemented the new process from the beginning. The product manager of this team was from Project A, the QA engineer from Project B. This made the adoption much easier. Also, many people in AdWords had heard about how I ran my projects and the barrier to try it out was considerably lower.

6.1. The world is better, but …

Overall, the more agile processes worked really well. Everybody noticed that the additional structure comes with very little overhead and fixes many of the issues that we had before.

We're still getting up to speed on the iteration-based development. It's been nice for development, now that our iterations are in sync w. AdWords code freeze cycle. It was hard at first for UI/PM, but has gotten easier as PM has assembled farther projecting roadmap, to give UI a clue what will be needed for a coming iteration.

Tech Lead

After a month or two, both product managers realized that they need to establish a requirement process that ensures that we not only implement little bits and pieces at a time but keep the overall release. This is an issue that I had with previous agile teams. I could persuade the product managers to dissect their releases into smaller chunks and prioritize them.

For these I created release burndown charts to track when they will be finished. At this point I started to measure progress on the release level in features complete. At this point it was very easy to convince the teams that this is the right measurement as it would give us a much better guidance where the release is.

The teams first thought that it was strange to have one iteration burndown chart and one release burndown chart. But after a few iterations they saw the benefit of both. The iteration burndown to guide actual development efforts. And the release burndown to guide the overall release planning.

An ongoing issue is the QA involvement. I constantly have to push the QA engineers to test features immediately after they are implemented. The reason is that the QA engineers support several projects. And the other projects are not agile, i.e. don't require much attention during development, but a lot at the end. This made it hard for the QA engineers to constantly spend a little bit of time each day on or project to give the engineers the immediate feedback. Right now, both teams question if it is worth the effort to include QA tasks and effort estimates in our planning as it does not seem to have any benefit.

For me, it seems like an extra task of updating a table with data (QA estimates) that's not of significance for me. But I'd really like to know if it's helpful to others. So far, most of the estimates have been 0.1 points.

QA Engineer

Finally, the teams do not try to create a releasable product at the end of the iteration (which is even harder because of the QA issue mentioned above). There are always tasks half implemented, not tested, need review… For now, I am not pushing too hard on this. The teams completely implement enough features per iteration that we can release those with the next AdWords update.

6.2. The project manager is dispensable

Recently, I went on a 3 week vacation. I was concerned how the teams would continue with the agile process during my absence and reminders and reinforcements of our agile practices.

But it turns out that the teams embraced the process enough to continue it even without any reinforcement. Iteration planning meetings happened, backlogs were created according to previous velocity, and daily standup meetings took place …

7. Where are we going from here

With the success of three project teams we are now prepared to make much bolder steps. Everybody in AdWords had at least heard about the advantages of the agile approach. Resistance at this point will be much less.

- Establish backlogs and burndown charts as status reporting standards in AdWords. Even if teams do not adopt other agile practices, these practices are easy to implement and provide a very good visibility for the teams themselves but also management and other outsiders.
- Other managers voiced interest. With a shadowing approach I will guide them through the agile process and try to give them enough experience to implement agile practices in their projects by themselves
- A few projects involve teams from several AdWords departments (frontend, backend, NetAPI…). Such projects always required much more management attention. As great as Google engineers and tech leads are, coordinating and synchronizing a teams efforts with other teams often distracts tech leads too much. We will either try to coordinate these teams as one big team (one backlog, one burndown chart) or use the "Scrum-of-Scrum" [vii] approach.
- During the first months at Google I heard from other departments who are using some agile practices or full-on Scrum/XP processes. To support this effort we started a grouplet[6] that focuses on agile development. We just recently started this grouplet and the initial response / interest was overwhelming – not only from engineering, but also other groups (QA, Product Management, UI, Usability)
- The UI development and usability part of our development projects is still very frontloaded. Almost all of this work is done before development starts. A few usability experts and UI designers showed interest in making this also part of the iteration-based development.

8. Summary

With the help of an experienced agile leader (ScrumMaster, XP coach…) it was possible to carefully introduce agile practices into Google - an environment that does not have an affinity to processes in general. Instead of introducing a grand new process, individual practices could be introduced either to fix observed issues or just to "try them out" – the development teams realized the advantages very soon.

Along with these practices came a visibility into the development status that gave the approach great management support.

[6] Google grouplets are cross-department groups which focus on a specific area of the software development process (there is a tech documentation grouplet, a build tools grouplet…) The members of the grouplet use their 20% time for their participation.

All this could be done without destroying the great bottom-up culture that Google prides itself of. The practices only affect how the projects are structured. Design and implementation remains fully an engineering responsibility. With some modifications, we could even keep the very strong role of tech leads and UI designers.

In keeping the great culture and self-organization of the teams, I could easily manage several projects in parallel. I could continue to rely on all core team members to communicate effectively without introducing any heavy processes.

5. References

[1] H. Takeuchi and I. Nonaka, "The New New Product Development Game," *Harvard Business Review,* 1986.

[2] J. O. Coplien, "Borland Software Craftsmanship: A New Look at Process, Quality and Productivity," in *5th Annual Borland International Conference*, Orlando, FL, 1994.

[3] A. C. Kay, "The early history of Smalltalk," presented at the The second ACM SIGPLAN conference on History of programming languages (HOPL-II), 1993.

[4] J. Collins, *Good to Great: Why Some Companies Make the Leap... and Others Don't*: Collins, 2001.

[5] R. A. Brooks, "Intelligence without representation," *Artificial Intelligence,* vol. 47, pp. 139-159, 1991.

[6] C. G. Langton, "Life at the Edge of Chaos," in *Artificial Life II, SFI Studies in the Sciences of Complexity*, Held Feb 1990 in Sante Fe, NM, 1992, pp. 41-91.

[7] C. Jones, *Software assessments, benchmarks, and best practices*. Boston, Mass.: Addison Wesley, 2000.

[8] M. Poppendieck and T. Poppendieck, *Lean Software Development: An Implementation Guide*: Addison-Wesley, 2006.

[9] M. Poppendieck, "PatientKeeper, the only company in the world that is not thrashing.," J. Sutherland, Ed. Boston, MA, 2007.

[10] C. Alexander, *The Timeless Way of Building, Volume 1*, 1979.

[11] H. Takeuchi and I. Nonaka, *Hitotsubashi on Knowledge Management*. Singapore: John Wiley & Sons (Asia), 2004.

[12] M. Poppendieck, "A History of Lean: From Manufacturing to Software Development," in *JAOO Conference*, Aarhus, Denmark, 2005.

[13] J. K. Liker, *The Toyota way : 14 Management Principles from the World's Greatest Manufacturer*. New York: McGraw-Hill, 2004.

[14] W. D. Holford and M. Ebrahimi, "Honda: Approach to Innovation in Aerospace and Automotive/Pick-Up Truck Development: A Dialectical Yet Coherent Firm," in *40th Annual Hawaii International Conference on System Sciences (HICSS-40)*, Big Island, Hawaii, 2007.

[15] P. M. Senge, *The Fifth Discipline: the Art and Practice of the Learning Organization*. New York: Currency, 1990.

[16] K. Schwaber, *Agile project management with Scrum*. Redmond, Wash.: Microsoft Press, 2004.

[17] K. Schwaber and M. Beedle, *Agile software development with scrum*: Prentice Hall, 2002.

[18] J. Sutherland*, et al.*, "Distributed Scrum: Agile Project Management with Outsourced Development Teams," presented at the HICSS'40, Hawaii International Conference on Software Systems, Big Island, Hawaii, 2007.

[19] J. Sutherland*, et al.*, "Scrum and CMMI Level 5: A Magic Potion for Code Warriors!," in *Agile 2007*, Washington, D.C., 2007.

[20] M. Striebeck, "Ssh! We're Adding a Process..." presented at the Agile 2006, Minneapolis, 2006.

[21] J. Sutherland, "Agile Development: Lessons Learned from the First Scrum," *Cutter Agile Project Management Advisory Service: Executive Update,* vol. 5, pp. 1-4, 2004.

[22] K. Schwaber, "Scrum Development Process," in *OOPSLA Business Object Design and Implementation Workshop*, J. Sutherland*, et al.*, Eds., London: Springer, 1997.
[23] K. Beck, *Extreme Programming Explained: Embrace Change*. Boston: Addison-Wesley, 1999.
[24] K. Beck, "Request for SCRUM Information," J. Sutherland, Ed. Boston: Compuserve, 1995.
[25] M. Beedle*, et al.*, "Scrum: A Pattern Language for Hyperproductive Software Development," in *Pattern Languages of Program Design*. vol. 4, N. Harrison, Ed., Boston: Addison-Wesley, 1999, pp. 637-651.
[26] G. Benefield, "Rolling Out Agile at a Large Enterprise," in *HICSS'41, Hawaii International Conference on Software Systems*, Big Island, Hawaii, 2008.
[27] K. Schwaber, *The Enterprise and Scrum*. Redmond, WA: Microsoft Press, 2007.
[28] J. Sutherland*, et al.*, "Adaptive Engineering of Large Software Projects with Distributed/Outsourced Teams," presented at the International Conference on Complex Systems, Boston, MA, USA, 2006.
[29] C. Jones, *Applied software measurement : assuring productivity and quality*, 2nd ed. New York: McGraw-Hill, 1997.
[30] M. Cohn, *Agile Estimation and Planning*: Addison-Wesley, 2005.
[31] P. DeGrace and L. H. Stahl, *Wicked problems, righteous solutions : a catalogue of modern software engineering paradigms*. Englewood Cliffs, N.J.: Yourdon Press, 1990.
[32] F. P. Brooks, *The Mythical Man Month: Essays on Software Engineering*: Addison-Wesley, 1995.
[33] W. A. Wood and W. L. Kleb, "Exploring XP for Scientific Research," *IEEE Software,* vol. 20, pp. 30-36, May/June 2003.
[34] C. Larman, *Agile & Iterative Development: A Manager's Guide*. Boston: Addison-Wesley, 2004.
[35] S. J. Gould, *The structure of evolutionary theory*. Cambridge, Mass.: Belknap Press of Harvard University Press, 2002.
[36] M. Fowler, "Is Design Dead?," *Software Development,* vol. 9, April 2001.
[37] J. Sutherland*, et al.*, *Business Object Design and Implementation*: Springer, 1997.
[38] K. Schwaber. (1995, 6 December 2005). *Scrum Development Process*. Available: http://jeffsutherland.com/oopsla/schwapub.pdf
[39] J. Sutherland, "Agile Can Scale: Inventing and Reinventing Scrum in Five Companies," *Cutter IT Journal,* vol. 14, pp. 5-11, 2001.
[40] J. Sutherland, "Future of Scrum: Parallel Pipelining of Sprints in Complex Projects," presented at the AGILE 2005 Conference, Denver, CO, 2005.
[41] F. P. Brooks, "No silver bullet - essence and accidents of software engineering," *Computer,* vol. 20, pp. 10-19, April 1987.
[42] G. Booch, *Object-Oriented Analysis and Design with Applications*: Benjamin Cummings Publishing Company, 1994.
[43] B. A. Ogunnaike and W. H. Ray, *Process Dynamics, Modeling, and Control*: Oxford University Press, 1994.
[44] J. Sutherland. (1996). *ScrumWeb Home Page: A Guide to the Scrum Development Process*. Available: http://jeffsutherland.com/scrum/original.html
[45] K. Schwaber, "Controlled Chaos: Living on the Edge," *American Programmer,* April 1996.

[46] AberdeenGroup, "Upgrading to ISV Methodology for Enterprise Application Development," *Product Viewpoint,* vol. 8, December 7 1995.
[47] M. Pittman, "Lessons Learned in Managing Object-Oriented Development," *IEEE Software,* vol. 10, pp. 43-53, Jan/Feb 1993.
[48] G. Booch, *Object Solutions: Managing the Object-Oriented Project*: Addison-Wesley, 1995.
[49] I. Graham, *Migrating to Object Technology*: Addison-Wesley, 1994.
[50] B. W. Boehm, "A Spiral Model of Software Development and Enhancement," in *International Workshop on Software Process and Software Environments*, Coto de Caza, Trabuco Canyon, California, 1985.
[51] J. Rumbaugh, "What Is a Method?," *Object Journal,* vol. Oct, 1995.
[52] W. S. Humphrey, *Introduction to the Personal Software Process*: Addison Wesley, 1996.
[53] H. Ziv and D. Richardson, "The Uncertainty Principle in Software Engineering," in *submitted to Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, 1997.
[54] P. Wegner, "Why Interaction Is More Powerful Than Algorithms," *Communications of the ACM,* vol. 40, pp. 80-91, May 1997.
[55] B. Boehm, "Project Termination Doesn't Mean Project Failure," *IEEE Computer,* vol. 33, pp. 94-96, 2000.
[56] D. K. I. Sobek*, et al.*, "Toyota's Principles of Set-Based Concurrent Engineering," *Sloan Management Review,* vol. 40, pp. 67-83, 1999.
[57] M. Fowler and J. Highsmith, "The Agile Manifesto," *Dr. Dobbs,* July 13 2001.
[58] W. S. Humphrey, *A Discipline for Software Engineering*: Addison-Wesley, 1995.
[59] Matisse Software, "The Emergence of the Object-SQL Database," Mountain View, CA2003.
[60] W. D. Hillis, *The Connection Machine*. Cambridge, MA: MIT Press, 1985.
[61] S. Levy, *Artificial Life : A Report from the Frontier Where Computers Meet Biology*, 1st ed. New York: Vintage, Reprint edition, 1993.
[62] C. Jakobson, "The Magic Potion for Code Warriors! Maintaining CMMI Level 5 Certification With Scrum.," J. Sutherland, Ed. Aarhus, Denmark: Agile 2007 paper in preparation, 2006.
[63] K. E. Nidiffer and D. Dolan, "Evolving Distributed Project Management," *IEEE Software,* vol. 22, pp. 63-72, Sep/Oct 2005.
[64] R. Zanoni and J. L. N. Audy, "Projected Management Model for Physically Distributed Software Development Environment," in *HICSS'36*, Hawaii, 2003, p. 294.
[65] J. Barthelemy, "The Hidden Costs of Outsourcing," *MITSloan Management Review,* vol. 42, pp. 60-69, Spring 2001.
[66] B. Gorzig and A. Stephan, "Outsourcing and Firm-level Performance," German Institute of Economic ResearchOctober 2002.
[67] StandishGroup. (2003). *2003 Chaos Chronicles*. Available: http://www.standishgroup.com/press/article.php?id=2
[68] J. Sutherland, "Future of Scrum: Parallel Pipelining of Sprints in Complex Projects with Details on Scrum Type C Tools and Techniques," PatientKeeper, Inc., Brighton, MA, 2005.
[69] M. Cohn, *User Stories Applied : For Agile Software Development*: Addison-Wesley, 2004.

[70] C. Jones, "Programming Languages Table, Release 8.2," Software Productivity Research, Burlington, MA1996.
[71] M. C. Paulk, *et al.*, *The Capability Maturity Model ®: Guidelines for Improving the Software Process*. Boston: Addison-Wesley, 1995.
[72] M. B. Chrissis, *et al.*, *CMMI – guideline for process integration and product improvement*, 2002.
[73] D. R. Goldenson and D. L. Gibson, "Demonstrating the impacts and benefits of CMMI," *CrossTalk,* October 2003.
[74] D. D. o. Science, "Maturity of customer and suppliers in the public sector," 2006.
[75] M. C. Paulk, "Extreme Programming From a CMM Perspective," *IEEE Software,* vol. 34, 2001.
[76] M. C. Paulk, "Agile Methodologies and Process Discipline," *CrossTalk,* 2002.
[77] Surdu, "Army Simulation Program Balances Agile and Traditional Methods with Success," *CrossTalk,* 2006.
[78] P. E. McMahon, "Lessons Learned Using Agile Methods on Large Defense Contracts," *CrossTalk,* 2006.
[79] D. Kane and S. Ornburg, "Agile Development: Weed or Wildflower," *CrossTalk,* 2002.
[80] Krasner and Houston, "Using the Cost of Quality Approach for Software," *CrossTalk,* November 1998.
[81] M. Diaz and J. King, "How CMM Impacts Quality, Productivity, Rework, and the Bottom Line," *CrossTalk,* March 2002.
[82] J. V. Sutherland, "ScrumMaster Certification Syllabus." Boston: Scrum Alliance, 2007.
[83] M. K. Kulpa and K. A. Johnson, *Interpreting the CMMI: A Process Improvement Approach*. Boca Raton: Auerbach Publications, 2003.
[84] J. Rose, *et al.*, "The Industrial Age, the Knowledge Age, the Internet Age: Improving Software Management," Aalborg University2004.
[85] J. O. Coplien, "Personal issues caused over 50% of productivity losses in the ATT Bell Labs Pasteur Project analysis of over 200 case studies," J. Sutherland, Ed., Personal communication ed. Lynby, Denmark, 2006.
[86] T. Sulaiman, *et al.*, "AgileEVM - Earned Value Management in Scrum Projects," in *Agile 2006*, Minneapolis, 2006.
[87] NetObjectives. (2005, 9 Feb). *Managing the Work*. Available: http://netobjectives.com/resources/downloads/ManagingTheWork.pdf
[88] P. M. Senge, *The Fifth Discipline: The Art and Practice of the Learning Organization, Revised Edition*. New York: Doubleday, 2006.
[89] G. Rothfuss. (2003, 6 December). *Metrics for Agile Methods*. Available: http://greg.abstrakt.ch/archives/2003/11/metrics_for_agile_methods.html
[90] R. Martens. (2005, 28 November). *On Be(coming) Agile: SCRUM Gathering – Advanced SCRUM and Rally as an Agile Enterprise*. Available: http://theagileblog.net/2005/11/scrum_gathering_advanced_scrum_1.html
[91] T. Poppendieck, "Roots of poor management practices," J. Sutherland, Ed. Boston, MA, 2006.
[92] M. Fackler, "The "Toyota Way" is Translated for a New Generation of Foreign Managers," in *New York Times*. New York, 2007.
[93] D. K. Taft. (2005). *Microsoft Lauds 'Scrum' Method for Software Projects*. Available: http://www.eweek.com/article2/0,1895,1885883,00.asp

[94] K. Johnson, *et al.*, "Scrum and CMMI Level 5: A Magic Potion for Code Warriors!," in *SEPG*, 2007.
[95] E. M. Goldratt and J. Cox, *The goal : a process of ongoing improvement*, 2nd rev. ed. Great Barrington, MA: North River Press, 1994.
[96] M. Cohen. (2006, Sep 4) Make a Bet on General Motors. *Forbes*. 162.
[97] A. MacCormack, *et al.*, "Exploring Tradeoffs Between Productivity and Quality in the Selection of Software Development Practices.," *IEEE Software,* pp. 78-85, 2003.
[98] I. Jacobson, *Object-Oriented Software Engineering: A Use Case Driven Approach*: Addison-Wesley, 1992.
[99] J. Sutherland and W.-J. van den Heuvel, "Enterprise Application Integration and Complex Adaptive Systems: Could System Integration and Cooperation be improved with Agentified Enterprise components?," *Communications of the ACM,* vol. 45, pp. 59-64, October 2002.
[100] J. M. Osier, *et al.*, "Keeping Track: Managing Messages with GNATS, The GNU Problem Report Managment System Version 4.0," Free Software Foundation2002.
[101] J. M. Bland and D. G. Altman, "Statistics Notes: Regression towards the mean," *BMJ,* vol. 308, pp. 1499-, June 4, 1994 1994.
[102] W. E. McCarthy, "The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment," *The Accounting Review,* vol. LVII, pp. 554-578, July 1982.
[103] G. Weinberg, *Quality Software Management Vol. 4: Anticipating Change*: Dorset House, 1997.
[104] J. H. Holland, *Hidden order : how adaptation builds complexity*. Reading, Mass.: Addison-Wesley, 1995.
[105] J. H. Holland, *Emergence : from chaos to order*. Reading, Mass.: Addison-Wesley, 1998.
[106] K. Kleinberg and T. Berg, "Mobile Healthcare: Applications, Vendors and Adoption," Gartner Group26 November 2002.
[107] S. Group, "There's Less Development Chaos Today," in *Software Development Times*, 2007.
[108] H. Kniberg, *Scrum and XP from the Trenches*. Stockholm: Crisp, 2007.
[109] I. o. Nonaka and H. Takeuchi, *The knowledge-creating company : how Japanese companies create the dynamics of innovation*. New York: Oxford University Press, 1995.
[110] E. M. Goldratt, *Theory of Constraints*. Burlington, MA: North River Press, 1990.
[111] W. Cunningham. (1999). *Wiki Wiki Web*. Available: http://c2.com/cgi/wiki
[112] J. Sutherland. (1997). *Jeff Sutherland's Scrum Log*.
[113] K. Schwaber. (1997). *Ken Schwabers's Scrum Web Page*.

Appendix I:

Scrum: A Pattern Language for Hyperproductive Software Development

Mike Beedle, Martine Devos, Yonat Sharon, Ken Schwaber, and Jeff Sutherland

Can a repeatable and defined process really exist for software development? Some think this is not only possible but necessary, for example, those who favor the CMM (Capability Maturity Model) approach to software development [71].

However, many of us doing work in the trenches have found over time that the *repeatable* or *defined* process approach makes many incorrect assumptions, such as the following:

- ***Repeatable/defined problem.*** A repeatable/defined process assumes that there is a step or steps to capture requirements , but in most cases, it is not possible to define the requirements of an application like that because they are either not well defined or they keep changing.
- ***Repeatable/defined solution.*** A repeatable/defined process assume that an architecture can be fully specified, but in reality it is evolved, partly because of missing or changing requirements (as described above), and partly because of the creative process involved in designing new software structures.
- ***Repeatable/defined developers.*** The capabilities of a software developer vary widely, so that a process that works for one developer may not work for another one.
- ***Repeatable/defined organizational Environment.*** Schedule pressure, priorities (e.g. quality vs. price vs. manpower), client behavior, and so on are never repeatable, and because they are highly subjective, they are very hard to define.

The problem with these assumptions is that these variables do have large variances. In real life projects there are always large dynamic variations that can have a great deal of impact on the overall project. For example, newly found changes in the requirements during an application implementation—a typical occurrence—may affect drastically a project's schedule that assumed that *all* the requirements would be captured up front. However, removing this uncertainty is nearly impossible because of the nearly universal sources of requirements change: business requirements driven changes, usability driven changes, re-prioritization driven changes, testing driven changes, and so forth. This issue cannot be solved through improved methods for identifying the user requirements. Instead it calls for a more complex process of generating fundamentally new operating alternatives.

In other words, once we accept that these dynamic variabilities do exist, we clearly need more adaptive ways to build software. However, what we fear is that most current methods do not allow us to build *soft enough* software: present methods and design paradigms seem to inhibit adaptability. Therefore the majority of software practitioners nowadays tend to become experts at what they can *specify in advance*, working with the unstated belief that there exists an optimal solution that can be planned a priori. Once technology is adopted by an organization, it oten

becomes a constraining structure that in part shapes the action space of the user. Thus we build software too much like we build hardware—as if it were difficult to change, as if it has to be difficult to change. In many organizations, "The system requires it" or the "System does not allow it" have become accepted (and often unavoidable) justifications for human behavior before and after the system is released to production.

In contrast, Scrum allows us to build *softer* software, so there is no need to write full requirements up fronts. Since the users do not know what is possible, they will ask for the pre-tech-paper solution that they perceive to be possible ("looking at the rearview mirror"). But in truth, not even the software developers know fully what can be built beforehand. Therefore, the user has no concept of what is possible before he or she can feel it or touch it. As such, The Scrum patterns presented here offer a collection of empirical techniques that assume up front the existence of uncertainty but that provide practical and specific techniques to tame it. These techniques are rooted in complexity management, that is, in self-organization, management of empirical processes, and knowledge creation.

In that sense, Scrum is not only a "parallel iterative and incremental" development method, it is also an "adaptive" software development method.

How does Scrum Work?

Scrum's goal is to deliver as much quality software as possible within a series (three to eight) of short time boxes (fixed-time intervals) called Sprints that typical last about a month.

Each stage in the development cycle (Requirements, Analysis, Design, Evolution, and Delivery) is now mapped to a Sprint or series of Sprints. The traditional software development stages are retained primarily for convenience tracking milestones. So, for example, the Requirements stage may use one Sprint, including the delivery of a prototype. The Analysis and Design stage may take one Sprint each, while the Evolution stage may take anywhere from three to five Sprints.

Editors Note: In recent years, release cycles have shortened to three months or less for most software products. Requirements are specified just enough and just in time to be ready at the start of the Sprint cycle. Sprints produce working software for review at the end of every Sprint. As a result, Analysis, Design, and Evolution occur in every Sprint. Sprint cycles in many companies have been shortened to two weeks or less. In the best companies, delivery is included in every Sprint [40].

Unlike a repeatable and defined process approach, in Scrum there is no predefined process within a Sprint. Instead, Scrum meetings drive the completion of the allocated activities.

Each sprint operates on a number of work items called a Backlog. As a rule, no more items are externally added into the Backlog within a Sprint. Internal items resulting form the original pre-allocated Backlog can be added to it. The goal of a sprint is to complete as much quality software as possible, but typically less software is delivered in practice.

The end result is that there are non-perfect releases delivered every Sprint.

During a Sprint, Scrum Meetings are held daily to determine the following:

- Items completed since the last Scrum meeting..
- Issues or blocks that need to be resolved. (The ScrumMaster is a team leader role responsible for resolving the blocks.)
- New assignments the team should complete before the next Scrum meeting.

Scrum Meetings allow the development team to "socialize the team members' knowledge" as well as produce a deep cultural transcendence. This "knowledge socialization" promotes a self-organized team structure within which the development process evolves on a daily basis.

At the end of each Sprint there is a demonstration to:

- Show the customer what's going on.
- Give the developer a sense of accomplishment.
- Integrate and test a reasonable portion of the software being developed.
- Ensure *real progress*, that is, the reduction of Backlog, not just the production of more paper/hours spent.

After gathering and reprioritizing leftover and new tasks, a new Backlog is formed and a new Sprint starts. Potentially, many other organization and process patterns may be used in combination with the Scrum patterns.

Figure: *The Scrum pattern language*

The Patterns

Sprint

Context

You are a software developer or a coach managing a software development team where there is a high percentage of discovery, creativity, or testing involved.

You are building or expanding systems, which allow partitioning of work, with clean interfacing, components, or objects.

Problem

You want to balance the needs of developers to work undisturbed and the need of management and the customer to see real progress, as well as control the direction of that progress throughout the project.

Forces

- Developers need time to work undisturbed, but they need support for logistics; management and users need to be convinced that real progress is being made.
- Often, by the time systems are delivered, they are obsolete or require major changes. The problem is that input from the environment is collected mostly at the start of the project, while the user learns mostly by using the system or intermediate releases.
- It is often assumed that the development process is a well-understood approach that can be planned and estimated. If a project fails, that is considered proof that the development process needs more rigor. These step by step approaches, however, don't work because they do not cope with the unpredictabilties, both human and technical, in system development. Therefore, at the beginning of a project it is impossible to make a complete, detailed specification, plan, or schedule because of the many uncertainties involved.
- Overhead is often created to prove that a process is on track. Current process automation adds administrative work for managers and developers and often results in marginally used development processes that become disk-ware. (Misfit: Activity is not synonymous with results. More often that not, a project plan shows activities but fails to ensure real progress or results.

Solution

Divide the project in Sprints. A Sprint is a period of approximately 30 days in which an agreed amount of work will be performed to create a deliverable. Each Sprint takes a pre-allocated amount of work from the Backlog, and it is assigned to Sprints by priority and by approximation of what can be accomplished during the Sprint's length. In general, chunks of high cohesion and low coupling are selected—either horizontal or vertical "packets," that is, vertical or horizontal components.

As a rule, nothing is added externally to the allocated Sprint Backlog during the Sprint. External additions are only added to the global Backlog, but blocks (unresolved issues) resulting from the Sprint can be added to the allocated Sprint Backlog. A Sprint end with a demonstration (Demo After Sprint) of new functionality.

This gives the developers space to be creative, to learn by exploring the design space and by doing actual work. Undisturbed by outside interruptions, they are free to adapt their ways of working using opportunities and insights. At the same time, this keeps management and other project stakeholders confident by showing real progress instead of documents and reports produced as *proof* of progress.

The net result is that each sprint produces a visible and usable deliverable that is shown to the users at the demo (Demo After Sprint). An increment can be either intermediate of shippable, but it should stand on its own. The goal of a Sprint is to complete as much quality software as possible and to ensure real progress, not paper milestones as alibis.

Editors Note: This strategy had a huge impact on global software development. Iterations that demonstrate early working software in order to incorporate real-time user feedback have increased project success industry-wide from 16.2% in 1994 to 35% in 2006. This increased the industry-wide return on dollar invested in software from 25 cents in 1998 to 59 cents in 2006 for a compound annual growth rate of 24% [107].

Rationale

- The fact that no items are added to the Backlog externally allows development to progress "full speed ahead," without needing to think about changes in direction.
- The fact that developers are not "tested" during the Sprint is empowering.
- The ability to choose a process per Sprint is empowering and enables adaptation to changing circumstances (different developers, different project phases, more knowledge, etc.)
- Sprints are short; therefore, the problem of completing a Sprint is much simpler that that of completing a project. It is easier to take up this smaller challenge.
- Developers get feedback frequently (at the end of each Sprint). They can therefore feel their successes (and failures) without compromising the whole project.
- Management has full control—it can completely change direction at the end of each Sprint.
- The end users are deeply involved throughout the development of the application through the Demos after the Sprints, but they are not allowed to interfere with the day-to-day activities. Thus ownership and direction still belong to the users but without their constant interference.
- Project status is visible since the Sprint produces working code.

Known Uses

At Argo, the Flemish department of education, we have been using Sprints since January 1997 on a large number of end-user projects and for the development of a framework for database, document management, and workflow. The Backlog is divided in sprints that last about a month. At the end of each Sprint, a working Smalltalk image is delivered with integration of all current applications. The team meets daily in Scrum Meetings, and Backlog is re-prioritized after the Demo in a monthly meeting with the steering committee.

Resulting Context

The result is a high degree of "effective ownership" by the participants, including users who stay involved through the Demos and the prioritizing of the Backlog. "Effective ownership" in this case means both empowerment and the involvement of all the participants.

At the end of a Sprint, we have the best approximation of what was planned at the start. In a review session, the supervisors have the opportunity to change the planning for the future. The project is totally flexible at this point. Target, product, delivery date, and cost can be redefined.

With Scrum we get a large amount of post-planning flexibility (for both customer and developer).

It may become clear in the daily Scrum Meetings throughout the Sprint that some team members are losing time at non- or less productive tasks. Alternatively, it may also become clear that people need more time for their tasks than originally allocated by management. Developers may turn out to be less competent or experienced at the allocated task than assumed, or they may be involved in political or power struggles. The high visibility of scrum, however, allows us to deal with these problems. This is the strength of the Scrum method manifested through the Scrum Meetings and the Sprints.

Difficulties in grouping Backlog for a Sprint may indicate that priorities are not clear to management or to the customer.

Backlog

Context (From: Sprint)

You are connected to a software project or any other project that is chaotic in nature that needs information on what to do next.

Problem

What is the best way to organize the work to be done next and at any stage of the project?

Forces

Traditional planning methods like Pert and Gantt assume that you know in advance all the tasks, all their dependencies, all task durations, and all available resources. These assumptions are wrong if the project involves any learning, discovery, creativity, or adaptation.

Solution

Use a Backlog to organize the work of a Scrum team.

The Backlog is a prioritized list. The highest priority Backlog item will be worked on first, the lowest priority Backlog item will be worked on last. No feature, addition, or enhancement to a product is worth fighting over; it is simply either more important or less important at any time to the success and relevance of the product.

Backlog is the work to be performed on a product. Completion of the work will transform the product from its current form into its vision. But in Scrum, the Backlog evolves as the product and the environment in which it will be used evolves. The Backlog is dynamic, constantly changed by management to ensure that the product defined by completing the Backlog is the most appropriate, competitive, useful product possible.

There are many sources for the Backlog list. Product marketing adds work that will fulfill their vision of the product. Sales adds work that will generate new sales or extend the usefulness to the installed base. Technology adds work that will ensure the product uses the most innovative and productive technology. Development adds work to enhance product functions. Customer support adds work to correct underlying product defects.

Only one person prioritizes work. This person is responsible for meeting the product vision. The title usually is product manager or product marketing manager. If anyone wants the priority for work changed, they have to convince this person to change that priority. The highest priority Backlog has the most definition. It is also prioritized with an eye toward dependencies.

Depending on how quickly products are needed in the marketplace and the finances of the organization, one or more Scrum Teams work on a product's Backlog. As a Scrum Team is available (newly formed or just finished a Sprint) to work on the Backlog, the team meets with the product manager. Focusing on the highest priority Backlog, the team selects a subset of the Backlog the team believes it can complete within a Sprint iteration (30 days or less). In doing so, the Scrum Team may alter the Backlog priority by selecting a Backlog that is mutually supportive, that is, one that can be worked on at once more easily than by waiting. Examples are multiple work items that require developing a common module or interface and that make sense to include in one Sprint.

The team selects a cohesive group of top priority Backlog items that, once completed, will have reached an objective, or milestone. This is stated as the Sprint's objective. During the Sprint, the team is free to not do work as long as this objective is reached.

The team now decomposes the selected Backlog into tasks. These tasks are discrete pieces of work that various team members sign up to do. Tasks are performed to complete Backlog to reach the Sprint objective.

Resulting Context

Projec work is identified dynamically and prioritized according to:

1. The customer needs
2. What the team can do

Scrum Meetings

Context (From: Backlog)

You are a software developer or a coach managing a software development team where there is a high percentage of discovery, creativity, or testing involved. An example is a first time delivery where the problem has to be specified, an object model has to be created, or new or changing technologies are being used.

Activities such as scientific research, innovation, invention, architecture, engineering and a myriad of other business situations may also exhibit this behavior.

You may also be a "knowledge worker," an engineer, a writer, a research scientist, or an artist, or a coach or manager who is overseeing the activities of a team in these environments.

Problem

What is the best way to control an empirical and unpredictable process such as software development, scientific research, artistic projects, or innovative designs where it is hard to define the artifacts to be produced and the processes to achieve them?

Forces

Estimation

- Accurate estimation for activities involving discovery, creativity, or testing is difficult because it typically involves large variances, and because small differences in circumstances may cause significant differences in results. These uncertainties come in at least five flavors:
  1. Requirements are not well understood.
  2. Architectural dependencies are not easy to understand and are constantly changing.
  3. There may be unforeseen challenges with the technology. Even if the challenges are know in advance, their solutions and related effort are not known.

4. There may be bugs that are hard to resolve in the software; therefore, it is typical to see project estimates that are several orders of magnitude off. You can't "plan bugs," you can only plan bug handling and provide appropriate prevention schemes based on the possibility of unexpected bugs.

   Example: You Got the Wrong Number. In projects with new or changing requirements, a new architecture, new or changing technologies, and difficult bugs to weed out, it is typical to see project estimates that are off by several orders of magnitude.
5. On the other hand, estimation *is* important. One must be able to determine what are the future tasks within some time horizon and prepare resources in advance.

Planning

- Planning and reprioritizing tasks takes time. Involving workers in time planning meetings decreases productivity. Moreover, if the system is chaotic, no amount of planning can produce uncertainties.

  Example: Paralysis by Planning. Some projects that waste everyone's time in planning everything to an extreme detail but are never able to meet the plans.
- A plan that is too detailed become large and is hard to follow; the larger the plan is, the more errors it will contain (or at the very least the cost of verifying its correctness grows).

  Example: The Master Plan Is a Great Big Lie. Many projects that try to follow a master plan fall into the trap of actually believing their inaccuracies and often face disappointment when their expectations are not met.
- No planning at all increases uncertainty among team members and eventually damages morale.

  Example: Lost Vision. Projects that never schedule anything tend to lose control over their expectations. Without some schedule pressure no one will do anything, and worse, it will become difficult to integrate the different parts being worked on independently.

Tracking

- Too much monitoring wastes time and suffocates developers.

  Example: Measured to Death. Projects that waste everybody's time in tracking everything to an extreme detail but are never able to meet the plans. (You measured the tire pressure until all the air was out!)
- Tracking does not increase the certainty of indicators because of the chaotic nature of the system. In fact, trying to control normal variations of a system will cause wide oscillations of the system, rendering it more chaotic.
- Too much data is meaningless—the Needle in the Haystack Syndrome.
- Not enough monitoring leads to blocks and possible idle time between assignments.

  Example: What Happened Here? Projects that never track anything tend to lose control over what is being done. Eventually no one really knows what has been done.

Solution

To provide for accurate estimates, plans, and appropriate tracking, meet with the team members for a short time (~15 minutes) in a daily Scrum Meeting, where the only activity is asking each participant the following three questions:

1. What have you worked on since the last Scrum Meeting? The ScrumMaster logs the tasks that have been completed and those that remain undone.
2. What blocks, if any, have you found in performing your tasks within the last 24 hours? The ScrumMaster logs all blocks and later finds a way to resolve the blocks.
3. What will you be working on in the next 24 hours? The ScrumMaster helps the team members choose the appropriate tasks to work on with the help of the Architect. Because the tasks are scheduled on a 24-hour basis, the tasks are typically small (Small Assignments).

This will provide you with more accurate estimates, short-term plans, appropriate tracking, an correcting mechanisms to react to changes and adapt every 24 hours.

Scrum Meetings typically take place at the same time and place every day, so they also serve to build a strong culture. As such, Scrum meetings are rituals that enhance the socialization of status, issues, and plans for the team. The ScrumMaster leads the meetings and logs all the tasks from every member of the team into a global project Backlog. He also logs every block and resolves each block while the developers work on other assignments.

Editors note: The Scrum Board has emerged as a best practice for a team to manage their own tasks. Teams meet in front of the Board which has multiple columns. The first column has User Stories from the Product Backlog (features to be delivered) on large cards prioritized in order of business value. At the start of the Sprint, the tasks to be accomplished for a User Story are in the left column as small cards. Each day developers move tasks to an "In Progress" column, then to a "Validation" column, then to a "Done" column. Estimates are updated on tasks daily and the Burndown Chart can easily be calculated and posted to the board [108].

The blocks logged by the ScrumMaster are now known as the "Impediment List" which needs to be prioritized. The block which is the most critical constraint to system throughput should be at the top of the list and the ScrumMaster should work on that one first. Tuning a development project is similar to tuning a computer system. It may not be obvious where the critical constraint lies and careful analysis may be required. The main choke point must be found and fixed first. The development system as a whole should then be allowed to stabilize and measured. The next critical block after restabilization may be in an unexpected place. That should be fixed next. Fixing too many things at once generates waste by fixing constraints that have minimal impact on throughput. This uses critical resources to change things that do not dramatically improve velocity. It makes it difficult to clarify system dynamics and tires out and demotivates the team, management, and the company.

Scrum meetings not only schedule tasks for developers, but can and should schedule activities for everyone involved in the project, such as integration personnel dedicated to configuration management, architects, ScrumMasters, or a QA team.

Scrum Meetings allow knowledge workers to accomplish mid-term goals typically allocated in Sprints that last a month or less.

Scrum Meetings can also be held by self-directed teams. In that case, someone is designated as the scribe and logs the completed activities of the Backlog and the existing blocks. All activities from the Backlog and the blocks and then distributed among the team for resolution.

The format of the Backlog and the blocks can also vary, ranging from a list of items on a piece of paper, to software representations of it over the Internet/Intranet [17]. The Scrum Meeting's frequency can be adjusted and typically ranges between 2 and 48 hours.

These meetings are often held standing up. This ensures that the meetings are kept short and to the point.

Rationale

It is very easy to over- or under-estimate, which leads either to idle developer time or to delays in completion of an assignment. Therefore, it is better to frequently *sample* the status of small assignments. Projects with a high degree of unpredictability cannot use traditional project planning techniques such as Gantt or PERT charts *only*, because the rate of change of what is being analyzed, accomplished, or created is too high. Instead, constant reprioritization of tasks offers an adaptive mechanism that provides sampling of systemic knowledge over short periods of time.

Scrum Meetings help also in the creation of an "anticipating culture" [103] because they encourage these productive values:

- They increase the overall sense of urgency.
- They promote the sharing of knowledge.
- They encourage dense communications.
- They facilitate honesty among developers since everyone has to give a daily status.

This same mechanism encourages team members to socialize, externalize, internalize, and combine technical knowledge on an ongoing basis, thus allowing technical expertise to become community property for the community of practice [109]. Scrum Meetings are therefore rituals with deep cultural transcendence. Meeting at the same place at the same time, and with the same people, enhances a feeling of belonging and creates the habit of sharing knowledge.

Seen from the System Dynamics point of view [88], software development has a scheduling problem because the nature of programming assignments is rather probabilistic. Estimates are hard to come by because:

- Inexperienced developers, managers, and architects are involved in making the estimates.
- There are typically interlocking architectural dependencies that are hard to manage.
- There are unknown or poorly documented requirements.
- There are unforeseen technical challenges.

As a consequence, the software development becomes a chaotic *beer game*, where it is hard to estimate and control the *inventory* of available developer's time, unless increased monitoring of small assignments is implemented [88, 110]. In that sense the Scrum Meeting becomes the equivalent of the thermometer that constantly samples the team's temperature.

From the Complexity Theory perspective [104, 105], Scrum allows flocking by forcing a faster agent interaction, therefore accelerating the process of self-organization because it shifts resources opportunistically through the daily Scrum Meetings.

This is understandable, because the relaxation of the self-organized multi-agent system is proportional to the average exchange among agents per unit of time. And in fact, the "interaction

rate" is one of the levers one can push to control "emergent" behavior—it is like adding an enzyme or catalyst to a chemical reaction.

In Scrum this means increasings the frequency of the Scrum Meetings, and allowing more *hyperlinks* as described below, but up to an optimal upper-frequency bound on the Scrum Meetings (meetings/time), and up to an optimal upper bound on the hyperlinks or the Scrum Team members. Otherwise the organization spends too much time socializing knowledge, instead of performing tasks.

Known Uses

(Mike Beedle) At Nike Securities in Chicago we have been using Scrum Meetings since February 1997 to run all of our projects including BPR and software development. Everyone involved in these projects receives a week of training in Scrum techniques.

(Yonat Sharon) At Elementrix Technologies we had a project that was running way late after about five months of development. Only a small part (about 20 percent) was completed, and even this part had too many bugs. The project manager started running bi-daily short status meetings (none of us was familiar with the term Scrum back then). In the following month, the entire project was completed and the quality had risen sharply. Two weeks later, a beta version was out. The meetings were discontinued, and the project hardly progressed since. I don't think the success of the project can be attributed to the Scrum Meetings alone, but they did have a big part in this achievement.

One of my software team leaders at Rafael implemented a variation of Scrum Meetings. He would visit each developer once a day, and ask him the three questions; he also managed a Backlog. This does not have the team building effects, but it does provide the frequent sampling.

C3 and Vcaps projects (described on wiki [111]) also do this. (BTW, I adopted this name in Hebrew, since in Hebrew "meeting" is "sitting," and so we say "standup sitting".)

Resulting Context

The application of this pattern leads to:

- Highly visible project status
- Highly visible individual productivity
- Less time wasted because of blocks
- Less time wasted because of waiting for someone else
- Increased team socialization

Conclusion

Scrum is a knowledge creating process with a high level of information sharing during the whole cycle and work progress.

The key to Scrum is pinning down the date at which we want completion for production or release, prioritizing functionality, identifying available resources, and making major decisions about architecture. Compared to more traditional methodologies, the planning phase is kept short since we know that events will require changes to initial plans and methods. Scrum uses an empirical approach to development where interaction with the environment is not only allowed but encouraged. Changing scope, technology, and functionality are expected; and continuous information sharing and feedback keeps performance and trust high.

Its application also generates a strong culture with well-defined roles and relationships, with meaningful and transcending rituals.

Acknowledgements

We would like to thank all of the Scrum users and reviewers from whom we have received feedback over the years. Also, we thank all of the members of the Chicago Patterns Group that attended an early review session of the Scrum Meeting pattern (especially Brad Appleton, Joe Seda, and Bob Haugen). Finally we thank our PloP'98 shepherd, Linda rising, for providing us comments and guidance to make our paper better.

(Personal acknowledgement from Mike Beedle.) I'd like to thank both Jeff Sutherland [112] and Ken Schwaber [113] for adapting the Scrum techniques to software in the early 1990s, and for sharing their findings with me. Scrum has made a significant contribution to the software projects in which I used the technique.

Index

[i] Controlchaos website - http://www.controlchaos.com/about/burndown.php

[ii] Mike Cohn, "Agile Estimating and Planning", pp. 35-42.

[iii] http://www.retrospectives.com/pages/whatIsARetrospective.html

[iv] XP.org website - http://www.extremeprogramming.org/rules/standupmeeting.html

[v] http://www.satisfice.com/articles/what_is_et.htm

[vi] Controlchaos website - http://www.controlchaos.com

[vii] Mountain goat website - http://www.mountaingoatsoftware.com/Scrum/Scrumteam.php