Inventing and Reinventing SCRUM in Five Companies

Dr. Jeff Sutherland

CTO, PatientKeeper, Inc.

21 September 2001

Introduction

In recent months, a wide range of publications—Software Development, IEEE Software, Cutter IT Journal, Software Testing and Quality Engineering, and even The Economist—have published articles on agile software development methodologies, reflecting a growing interest in these new approaches to software development (Extreme Programming, Crystal Methodologies, SCRUM, Adaptive Software Development, Feature-Driven Development and Dynamic Systems Development Methodology among them). In addition to these "named" methodologies, scores of organizations have developed their own "lighter" approach to building software. The formation of the Agile Alliance by a group of expert consultants and authors on development process has fueled increasing interest in ways to deliver quality software in short, fixed delivery schedules, under severe time to market pressures (Fowler and Highsmith 2001).

The goal of SCRUM is to deliver as much quality software as possible within a series of short time boxes called Sprints that last about a month. It is characterized by short, intensive, daily meetings of every person on a software delivery team, usually including product marketing, software analysts, designers, and coders, and even deployment and support staff. SCRUM project planning uses lightweight techniques such as Burndown Charts, as opposed to PERT charts. A PERT chart is only as good as the assumptions inherent in the critical path represented on the chart. In agile development, the critical path usually changes daily, rendering any given PERT chart obsolete within 24 hours. The solution is using a technique to calculate the velocity of development. The neural networks in the brains of team members are used on a daily basis to calculate the critical path. This allows the plan to be recalculated and the velocity of “burndown” of work to be computed. Team efforts to accelerate or decelerate the velocity of burndown allow a team to “fly” the project into a fixed delivery date.

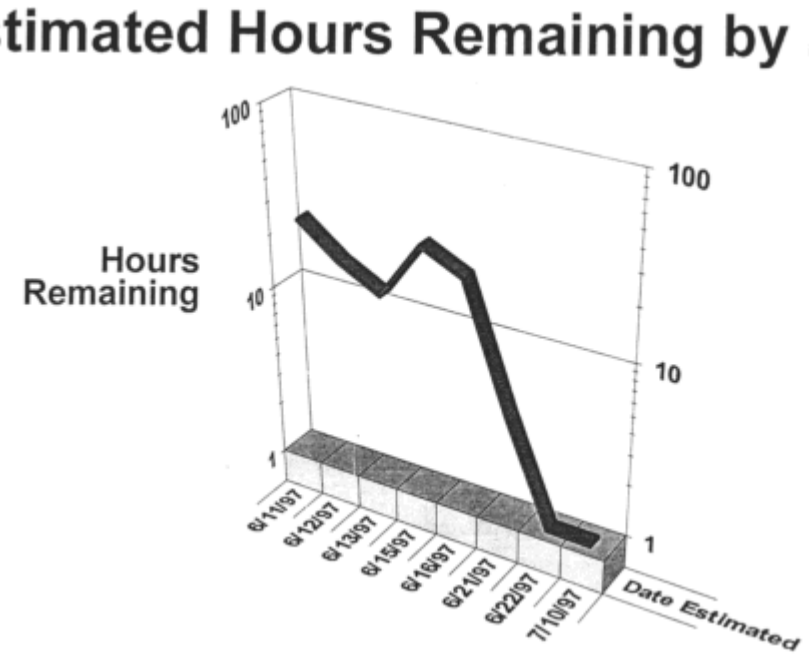

Figure 1: Burndown Chart – Advanced Development Methodologies, www.controlchaos.com

Details of the SCRUM approach have been carefully documented elsewhere. SCRUM is the only agile methodology that has been formalized and published as an organizational pattern for software development (Beedle, Devos et al. 1999). 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 which states that for a new software system, the requirements will not be completely known until after the users have used it. SCRUM allows for Ziv's Uncertainty Principle in software engineering - uncertainty is inherent and inevitable in software development processes and products (Ziv and Richardson 1997). And it support Wegner's Lemma which states that it is not possible to completely specify an interactive system (Wegner 1995). Most software systems built today are object-oriented implementations which depend on environmental inputs to determine process outputs, i.e. 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 (Boehm 2000).

The 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 processe that support 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 for software teams at Easel Corporation in 1993 where I was VP of Object Technology. We built the first object-oriented design and analysis tool that incorporated round-trip engineering in the initial SCRUM. A second SCRUM implemented the first product to completely automate object-relational mapping in an enterprise development environment. I was assisted by two world-class developers, Jeff McKenna, now an extreme programming (XP) consultant, and John 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 office.com). I asked Ken Schwaber, CEO of Advanced Development Methodologies, to help me incorporate SCRUM into Individual's development process. In the same year I took SCRUM to IDX Systems when I assumed the positions of Senior VP of Engineering and Product Development and CTO. IDX, one of the largest healthcare software companies, was the proving ground for multiple team SCRUM implementations. At one point, almost 600 developers were working on dozens of products. In 2000, SCRUM was introduced to PatientKeeper, a mobile/wireless healthcare platform company where I became CTO. So I have experienced SCRUM in five companies which varied widely in size. They were proving grounds for SCRUM in all phases of company growth, from startup, to initial IPO, to mid-size and then a large company delivering enterprise systems to the marketplace.

There were some key factors that influenced the introduction of SCRUM at Easel Corporation. The book "Wicked Problems, Righteous Solutions" (DeGrace and Stahl 1990) 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 user knows 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.

The team based "All-at-Once" model was based on the Japanese approach to new product development, Sashimi and SCRUM. We were already using production prototyping to build software. It was implemented in slices (Sashimi) where an entire piece of fully integrated functionality worked at the end of an iteration. What intrigued us was Takeuchi and Nonaka's description of the team building process in setting up and managing a SCRUM (Takeuchi and Nonaka 1986). The idea of building a self-empowered team where 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 resonated with the systems thinking approach being promoted by Senge at MIT (Senge 1990).

We were also impacted by recent publications in computer science. Peter Wegner at Brown University demonstrated that it was impossible to fully specify or test an interactive system which is designed to respond to external inputs, i.e. Wegner's Lemma (Wegner 1997). Here was mathematical proof that any process that assumed known inputs, like the waterfall method, was doomed to failure when building an object-oriented system. We were prodded into setting up the first SCRUM meeting after reading Coplien's paper on Borland's development of Quattro Pro for Windows. The Quattro team delivered one million lines of C++ code in 31 months with a 4 person staff growing to 8 people later in the project. This was about a 1000 lines of deliverable code per person per week, probably the most productive project ever documented. The team attained this level of productivity by intensive interaction in daily meetings with project management, product management, developers, documenters, and quality assurance staff.

Software Evolution and "Punctuated Equilibrium"

Our daily meetings at Easel were disciplined in the way we that we now understand as the SCRUM pattern (Beedle, Devos et al. 1999). 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.

It is well understood in biological evolution that change occurs sharply at intervals separated by long periods of apparent stagnation, leading to the concept of punctuated equilibrium (Dennett 1995). Computer simulations of this phenomenon suggest that periods of equilibrium are actually periods of ongoing genetic change of an organism. The effects of that change are not apparent until several subsystems evolve in parallel to the point where they can work together to produce a dramatic external effect (Levy 1992). This punctuated equilibrium effect has been observed by teams working in a component based environment with adequate business process engineering tools and the SCRUM development process accentuates the effect.

By having every member of the team see every day what every other team member was doing, we began to get comments from one developer that if he changed a few lines of code, he could eliminate days of work for another developer. This effect was so dramatic that the project accelerated to the point where IT HAD TO BE SLOWED DOWN. This hyperproductive state was seen in a several subsequent SCRUMs but never so dramatic as the one at Easel. It was a combination of the skill of the team, the flexibility of a Smalltalk development environment, and way we approached production prototypes that evolved into deliverable product.

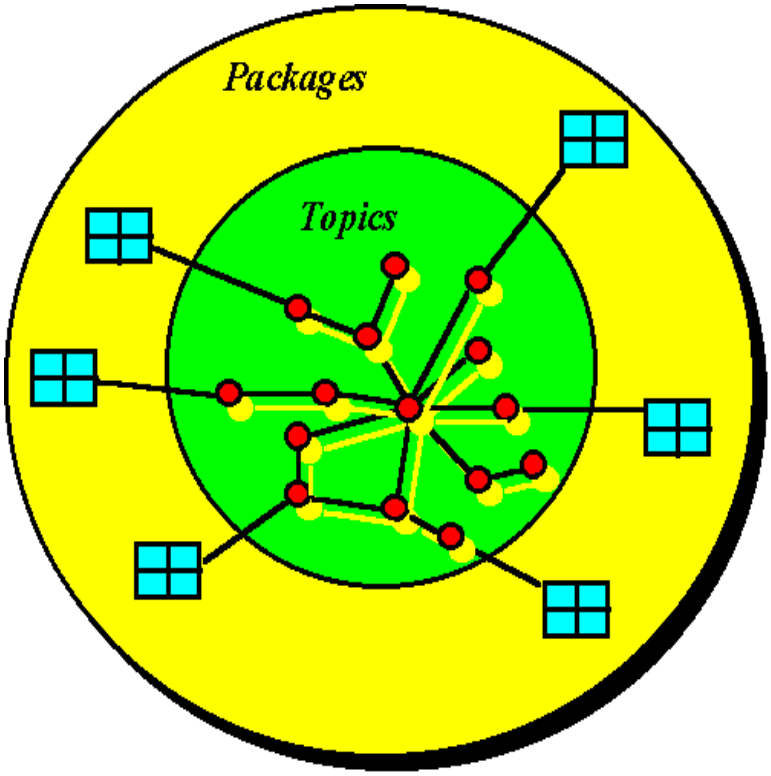

Figure 2: Initial SCRUM View of a Software System

The first SCRUM worked from a unique view of a 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. Topic areas are business object components. Changes are introduced into the system by introducing a unit of work that alters a component. The unit of work in the initial SCRUM was called a Synchstep.

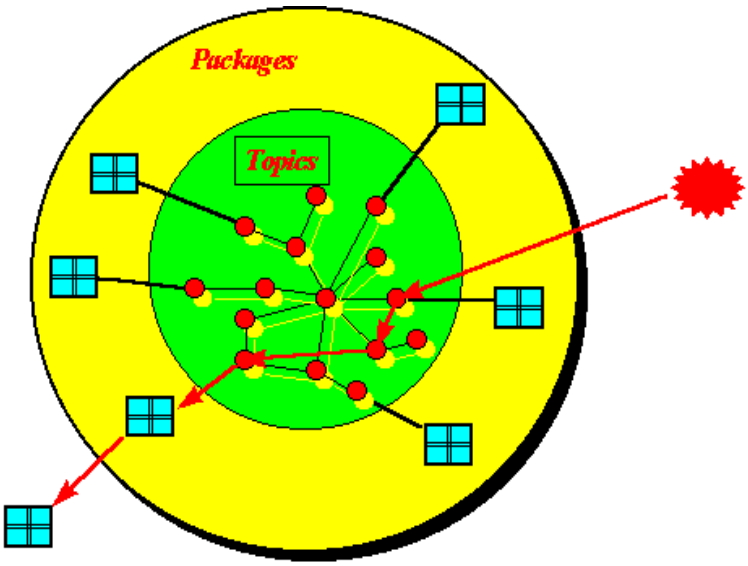

Figure 3: Firing a Synchstep

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

The key to entering a hyperproductive state was not just the SCRUM organizational pattern. We did constant component testing of topic areas, integration of packages, and refactoring of selected parts of the system. These activities have become key features of eXtreme Programming (Fowler 2001).

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 reminded me of the stories about the Celtics basketball team at their peak where they could do no wrong. The impact of entering the “zone” was not just hyperproductivity. Peoples personal lives were changed. People said they would never forget working on such a project and they would always be looking for another experience like it. It induced open, team oriented, fun loving behavior in unexpected persons and eliminated those who were not productive from the team through peer embarrassment.

VMARK – the first Senior Management SCRUM

When Easel Corporation was acquired by VMARK (now Informix), the original SCRUM team continued their work on the same product. The VMARK senior management team was intrigued by SCRUM and asked me to run a senior management team SCRUM weekly to drive all the companies products to the Internet. These meetings started in 1995 and within a few months, the team had caused the introduction of two new Internet products and repositioned 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.

INDIVIDUAL – the first Internet SCRUM

In the spring of 1996, I returned to Individual, Inc., a company I cofounded as VP of Engineering. Much of the SCRUM experience at Individual has been documented by Ken Schwaber (Schwaber and Beedle 2001). The most impressive thing to me about SCRUM at Individual was not that the team delivered two new Internet products in a single quarter, and multiple releases of one of the products. It was the fact that SCRUM eliminated about 8 hours a week of senior management meeting time starting the day the SCRUM began. Because the company had just gone public at the beginning of the Internet explosion, there were multiple competing priorities and constant revision of market strategy. As a result, the development team was constantly changing priorities and unable to deliver product. The management team was meeting almost daily to determine status of priorities that were viewed differently by every manager.

The solution was to force all decisions to occur in the daily SCRUM meeting. If anyone wanted any status or wanted to influence any priority, they could only do it in the SCRUM. I remember 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 immediate respond to her despair. Over a period of two weeks, the team self-organized around a plan to meet her priorities with achievable technical delivery dates. When she agreed to the plan, she no longer had to attend any SCRUM or status meetings. The SCRUM reported status on the web with green lights, yellow lights, and red lights for pieces of functionality. In this way the entire company knew status in real time, all the time.

IDX Systems – the first SCRUM in the Large

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

The approach at IDX was to turn the entire development organization into an interlocking set of 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 is 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 where only able to meet the industry average in function points per month delivered, several teams moved into the hyperproductive state producing deliverable functionality at 4-5 times the industry average. These teams became shining stars in the organization and examples for the rest of the organization to follow.

PatientKeeper SCRUM – Integration with eXtreme Programming

In early 2000, I joined PatientKeeper, Inc. as Chief Technology Officer and began introducing SCRUM into a startup company. I was the 21st employee and we grew the development team from a dozen people to 45 people in six months. PatientKeeper deploys mobile devices in healthcare institutions to capture and process financial and clinical data. Server technology synchronizes the mobile devices and moves data to and from multiple backend legacy systems. A 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 been around 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 programming. We have been able to do some team programming and constant testing and refactoring, particularly as we have migrated all development to Java and XML. It has been more difficult to introduce these ideas when developers are working in C and C++. After a year of SCRUM meetings in all areas of development, our processes are maturing enough to capitalize on SCRUM project management techniques which are now being automated.

Conclusions

After introducing SCRUM into five different companies with different sizes and different technologies, I can confidently say that SCRUM works in any environment and can scale into programming in the large. In all cases, it will radically improve communication and

delivery of working code. The next challenge for SCRUM, in my view, is to provide a tight integration of the SCRUM organization pattern and XP programming techniques. I believe this integration can generate a hyperproductive SCRUM on a predictable basis. The first SCRUM did this intuitively before XP was born and that was its key to extreme performance and life changing experience. In addition, the participation of SCRUM leaders in the Agile Alliance (Fowler and Highsmith 2001), a group which has absorbed all leaders of well known lightweight development processes, will facilitate wider use of SCRUM and its adoption as an enterprise standard development process.

Beedle, M., M. Devos, et al. (1999). SCRUM: A Pattern Language for Hyperproductive Software Development. Pattern Languages of Program Design 4. N. Harrison, B. Foote and H. Rohnert, Addison-Wesley.

Beedle, M., M. Devos, et al. (1999). SCRUM: A Pattern Language for Hyperproductive Software Development. Pattern Languages of Program Design. N. Harrison, Addison-Wesley. **4:** 637-651.

Boehm, B. (2000). "Project Termination Doesn't Mean Project Failure." IEEE Computer: 94-96.

DeGrace, P. and L. H. Stahl (1990). Wicked problems, righteous solutions : a catalogue of modern software engineering paradigms. Englewood Cliffs, N.J., Yourdon Press.

Dennett, D. C. (1995). Darwin's dangerous idea : evolution and the meanings of life. New York, Simon & Schuster.

Fowler, M. (2001). "Is Design Dead?" Software Development **9**(4).

Fowler, M. and J. Highsmith (2001). "The Agile Manifesto." Software Development **9**(8): 28-32.

Fowler, M. and J. Highsmith (2001). "The Agile Manifesto." Software Development: 28-32.

Levy, S. (1992). Artificial life : the quest for a new creation. New York, Pantheon Books.

Schwaber, K. and M. Beedle (2001). Agile Software Development with SCRUM, Prentice Hall.

Senge, P. M. (1990). The fifth discipline : the art and practice of the learning organization. New York, Doubleday/Currency.

Takeuchi, H. and I. Nonaka (1986). "The New New Product Development Game." Harvard Business Review(January-February).

Wegner, P. (1995). "Interactive Foundations of Object-Based Programming." IEEE Computer **28**(10): 70-72.

Wegner, P. (1997). "Why Interaction Is More Powerful Than Algorithms." Communications of the ACM **40**(5): 80-91.

Ziv, H. and D. Richardson (1997). The Uncertainty Principle in Software Engineering. ICSE'97, 19th International Conference on Software Engineering, Boston, MA, IEEE.