AGILE 2005 Conference
July 24-29, 2005
Marriott Denver City Center

Future of Scrum: Parallel Pipelining of Sprints in Complex Projects

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

Abstract

The Scrum Agile development process was invented to rapidly drive new product to market. Here, one of the inventors of Scrum goes back to Scrum basics, throws out preconceived notions, and designs Advanced Scrum using multiple overlapping Sprints within the same Scrum teams. This methodology delivers increasing application functionality to market at a pace that overwhelms competitors using 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. Administrative overhead for dozens of enterprise product releases a year is less than 60 seconds a day per developer and less than 10 minutes a day for a Scrum Master. While Advanced Scrum is not for the uninitiated, the future of Scrum is still Scrum, just faster, better, and cooler.

1. Scrum Evolution

Evolution occurs in dynamic response to environmental demands. Now that the Scrum community has over 2000 [redacted] Scrum Masters and tens of thousands of projects completed, 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 preconceptions).

One of the influences that sparked the creation of the Scrum Agile development process was a Harvard Business Review paper on Japanese new product development by Takeuchi and Nonaka [21]. 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 authors viewed Type A product development as implemented at NASA as an outmoded relay race process. Fuji-Xerox abandoned the NASA approach for Type B which they called "Sashimi" because slices of work overlapped with collaboration between phases. Type C was implemented at Canon and Honda. Takeuchi and Nonaka envisioned Types B and C as a Rugby approach where multiple phases of product development are 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.

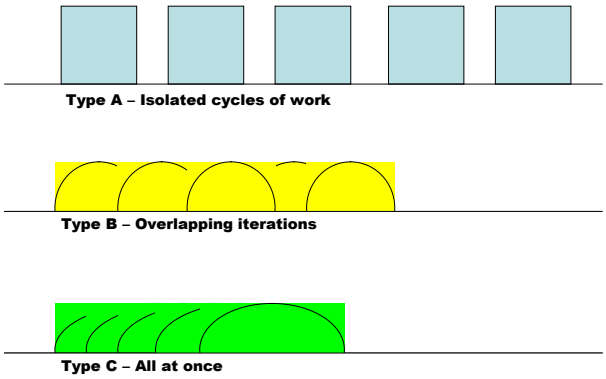

Figure 1: Type A, B, and C strategies for delivering product [20].

After discussing the notion of various types of Scru[redacted] Scrum [redacted] with development teams at Microsoft, Yahoo, Ariba, Adobe, GE Healthcare, and other companies, it appeared that the chart in Figure 1 can be applied to a higher level of thinking about three types of Scrum—going beyond the thinking of Takeuchi and Nonaka.

In a Type A Scrum, all development occurs in an increment within the time box of Scrum iteration called a Sprint. A side effect of this approach is downtime between iterations when reorganizing for the next Sprint. Nevertheless, 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 Type B Sprint allows work to flow smoothly from Sprint to Sprint. Product backlog requirements for the next Sprint are developed in the current Sprint. This has enabled some development organizations to deliver more working product than sales, marketing, or customers can absorb. The development bottleneck is eliminated and the company can adopt new strategies and create new products that were previously impossible to deliver.

Type C Sprint can be envisioned as overlapping Sprints by running software releases through the same Scrum team at the same time. 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. Competitors can be overwhelmed and market dominance achieved.

Takeuchi and Nonaka observed that 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 a dramatic increase in business opportunity.

2. Scrum Evolution in Practice

The evolution of Scrum in five companies from 1993-2001 has been described previously [16, 17]. Here we focus on continued evolution of Scrum theory using PatientKeeper, Inc., as an example. During 2000 we first automated a solution for Type B 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 2001, we began to solve the problem of multiple projects pipelined through the same team (or set of teams) and have been running a Type C Scrum for over four 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 were 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 sporadically and team programming was common where many programmers worked together around a table for the entire day. Daily Scrum of Scrums meetings became the norm. By 2003 we established weekly MetaScrum meetings of all company stakeholders to review scheduled release deliveries.

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. In 2004, more than 45 enterprise 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 Women's Hospitals) in Boston, Johns Hopkins in Baltimore, and Duke University Health System in North Carolina. These clients provide an excellent test bed for scalability of a Type C Scrum. They require a high level of product adoption by difficult and discriminating users (physicians), support for disparate wireless networks across an enterprise, integration with many clinical and financial systems in diverse IT infrastructures, and thorough testing and certification by the customer.

3. The First Scrum – Type A

Some corporations view Type A Scrum as useful for education and training on the pace of Scrum and particularly suited to new Scrum teams. Its downside is that it creates a loss of time between Sprints when the team is reorganizing for the next Sprint.

At Easel Corporation in 1993 we initially applied Type A 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 [18]. 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 loss of time was considered a problem 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.

In addition to loss of productivity between Sprints in a Type A Scrum, it takes time during the Sprint for developers to get enough clarity about the user requirements to start coding. It was often halfway through a Sprint before developers understood the user experience well enough to implement a solution. This created 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 phenomenon can cut Sprint productivity in half.

A Type A Sprint is sometimes used to pilot Scrum. It allows systematic application of the Scrum process with enough time to refine operations and regroup between Sprints. It forces all-at-once type thinking as everything has to happen for a specific Sprint within the time box of that Sprint. Initially, the benefits in training can overwhelm lost productivity and without the ability to execute a Type A Scrum well, it is not possible to effectively implement a more sophisticated process.

The benefits of Type A Scrum are:

- Total focus on iteration in process
- Ease of implementation
- Developing and understanding the pace of Scrum
- Clearly defined iterations

The problems with Type A Scrum are:

- Loss of time to market
- Disruption of pace of Scrum because of developer lack of understanding of the user experience
- Loss of productivity (and market share) due to resulting delays

4. Type B Scrum

The way to overcome loss of time to market with a Type A Scrum is to insert tasks in a current Sprint that stage work for a subsequent Sprint. A minimal specification of the user experience for a feature can be defined prior to the Sprint where it is implemented. This allows Sprints to be executed continuously with the Product Backlog defined and ready at the beginning of each Sprint.

The need to start development with adequate functional specifications was observed by MacCormack [12] 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.

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, consistent with Agile thinking. However, he found a strong correlation between adequate product specifications and productivity.

A functional specification that is complete enough for the next iteration to allow developers to begin work without false starts will enhance feature delivery within Sprints and improve throughput. Although 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 as a shortage of expert developers whose skills are not easily transferable. Constraint analysis shows mathematically that the biggest bottleneck should be eliminated first [7] (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.

A caveat is that Type B Scrum will not work in a company that has not implemented a sustainable development process. Scrum teams must 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 Sprints 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 is not an implementation of Scrum and makes it impossible for a team to enter the hyperproductive state for which Scrum was designed.

The key indicators that Scrum is working must be visible in a Type A Scrum before moving to Type B:

- Team autonomy – the Scrum team is (and feels) totally responsible for their product and no outside agency impacts the work plan of the team inside a Sprint. The Product Owner is part of the Scrum and helps with product design questions and implementation within a Sprint.
- 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 Scrum at all times without building a sustainable pace of development will negatively impact morale. On complex development projects, it can take a new engineer several months to come up to full productivity. Abdel-Hamid [3] shows through simulations of the software development microsystem that

under-resourcing increases total project cost by about 25%. If turnover is 20%, you are effectively under-resourcing by 20%. Extrapolating from Abdel-Hamid's data your development team productivity may be down 15% from this alone. The personnel churn may cause development task delay as specialized resources must be shifted to complete them, reducing productivity even more. If morale drives the pace of development down further, you may cut productivity in half due to Abdel-Hamid's "dynamic motivation factors."

Conversely, if Scrums are running well, pre-staging functional specifications in the right way in a Type B Scrum will eliminate false starts within a Sprint and downtime between Sprints. This has more than doubled productivity for some experienced Scrum teams. In companies seeking to expand market share and dominate a market segment, this advantage is absolutely compelling.

4.1 Staging Functional Specifications for a Type B Sprint

Maintaining the agility of the Scrum process requires a minimalist approach to functional specification. A minimal amount of documentation for a product feature may be a few pages and definitely not hundreds of pages, just enough documentation so that engineers understand the user experience.

At PatientKeeper a product specification needs screen shots, data requirements, workflow from screen to screen, and business logic that must be executed. This has been essential because all new functionality must be prototyped and tested by physician users before implementation. The minimum documentation required to achieve Jacobsen's overview of an object-oriented analysis of the problem [10] has been an excellent guideline. At PatientKeeper we have well educated physicians that often serve as Product Owners. While some of them have no formal training in software development, they quickly learn how to elaborate use cases in a way that define the user experience for a physician using PatientKeeper products. In addition, these Product Owners are action oriented, knowledgeable users, and strongly resistant to analysis paralysis. They avoid time spent on excess documentation, making them excellent Agile Product Owners 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. Members of the development team work with the Product Owner from the beginning of requirements creation. In the worst case, this might require 25% of Scrum development resources during a sprint. However, it avoids the 25% lag time between sprints. So you at least break even on resource allocation.

The real gain from a Type B Scrum is having the Product backlog fully loaded, prioritized, and ready for breakdown into Sprint backlog tasks 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 Scrum Master is the leader of a Scrum team, manages the project, and in some cases is a strong technical contributor. The Product Owner and Scrum Master are continuously working to peel items off the Product backlog, initialize breakdown of Product Backlog items into Sprint tasks, and assign tasks to a developer's queue. The developer decides how to order the work or, in some cases, refines the granularity of tasks and assigns them appropriate team members.

4.2 Product Owner as Part of the Scrum Team

The original Japanese view of a product development Scrum created a cross-functional team that was totally responsible for the product [21]. 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 [16, 17].

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 Scrum Master 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 Scrum Master.

The best way to visualize Scrum responsibilities is to think of a Scrum team as analogous to a high performance car in a rally race. The Product Owner is the navigator and the Scrum Master is the driver. The team is the engine, the chassis, the drive train, and the wheels. The Scrum Master 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 there is a demo where other players can suggest modifications to improve production in the next Sprint.

4.3 Type B Scrum Enables Hyperproductive Teams

Giving the Product Owner accountability for the Sprint backlog builds strong Product Owners with hands on control of the product feature and function. It conditions the development team to move rapidly towards the goal without analysis paralysis. A combination of a forceful driver coupled to a strong navigator and a high performance and reliable car wins the race. 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 on a development team that can deliver more product than anyone can absorb is exhilarating and allows the team to focus on higher goals like being the best product of its class in the industry.

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.

5. Evolution of Type C Scrum

Scrum is an organizational pattern [4] that is designed for control of an activity that is highly unpredictable. 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 by building a software framework and open application programming interfaces (APIs) that would allow integration with many development partners on both the backend and the frontend. A web services platform with a services oriented architecture was selected.

In addition to a server architecture which was Java/XML based, 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 similar 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 deployments in large enterprises on a monthly basis, forced a new type of Scrum to be implemented at PatientKeeper.

5.1 Case Study Context

PatientKeeper builds a software platform that takes information from disparate clinical, financial, and scheduling 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 is stored in a clinical data repository, often a third party system.
- Some or all data is forward-cached in a PatientKeeper clinical repository.
- Massive multi-threading of data requests to repositories with extensive use of in-memory cache improves performance.
- On a handheld device, a specific physician's data is stored locally.

Software and data must be consistent across four tiers at all times. This forced PatientKeeper to do totally integrated builds multiple times per day to assure that software in all four tiers of the architecture worked consistently. Developers work off the latest build in a code branch and on the latest code branch whenever possible. Quality Assurance has to validate that all architectural layers work together to provide consistent data to the end user for every release.

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. Tightly coupling of these teams in a daily Scrum of Scrums meeting assures that all software is interoperable all the time.

5.2 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 available 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 was used as a strategic weapon.

The customer based rapidly evolved to multiple hospital systems to be installed each month. 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. An integrated, enhanced release for new site requirements was required on a monthly basis.

The monthly deployment of new software releases into new enterprise sites required rapid bug and feature fixes for unanticipated implementation issues. This drove PatientKeeper to weekly maintenance releases. In addition, the PatientKeeper product roadmap required ongoing delivery of entire new applications to the market. The right timing for new applications was quarterly major product releases.

5.3 Case Study Forces

Resource constraints forced every developer to be focused 100% on building the system. Scrum Masters 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. They demanded 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 Scrum Master 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?
- Weekly and monthly releases would be packaged into Sprints that always released production code to large enterprises at the end of the Sprint. There are 12 months and 52 weeks in a year, a possible 64 releases a year to be managed with high quality. How was that possible?

5.4 Type C Scrum Solution

The Type C Scrum Solution required several innovations that affected all parts of the company. In effect, the company became a Scrum company with all activities tied to an automated data system that reflected release planning and Scrum implementation, as well as installation, support team, and customer feedback.

- A MetaScrum was created to allow company leadership to manage multiple simultaneous product releases.
- Daily Scrum of Scrums became the major Scrum meeting.
- Quality Assurance was reorganized
- Build process ran more frequently and needed to be more robust and easier to fine tune.
- Regression testing automation improved significantly.
- New tools for automated data collection and reporting were developed.
- The company became totally transparent. All data was available to everyone in real time all the time.

Here we describe the company infrastructure required to run a successful Type C Scrum. Many details of tools and techniques within the development team are beyond the scope of this paper and described elsewhere [19]

5.4.1 MetaScrum

Managing multiple simultaneous releases of software requires regular review and fine tuning of release schedules. Since every Sprint resulted in a production release of software, Sprints must be carefully coordinated.

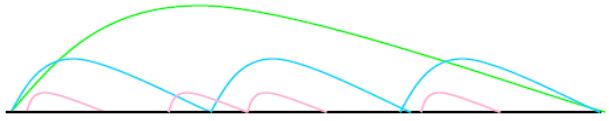

Figure 2: Simultaneous overlapping Sprints running through a single set of development teams [6].

Weekly Sprints are maintenance fixes or minor enhancements typically generated by issues preventing a customer from going live or causing the system to fail during production. Although these are highly impacted by changing requirements, they are scheduled and committed to customers on fixed dates.

Monthly releases are targeted as a set of customer go-live dates that required specific enhancements for each customer. If a new customer is added to the queue or a customer drops out of the queue, the release might have to be reorganized.

Quarterly releases of major functionality can only be completed when weekly and monthly releases are on

schedule. Priorities might change during a quarter because of market changes or new customer priorities. A large partner such as Cerner or GE Healthcare could highlight new demands, request acceleration of functionality, or cause delays.

To deal with simultaneous weekly and monthly Sprints, along with quarterly major releases, a MetaScrum was formed at PatientKeeper led by the lead Product Owner. This is a weekly meeting that typical takes 1.5 hours and includes the CEO and other senior management, as well as leadership from marketing, development, quality assurance, installation, and support.

Each release is reviewed. Sprints may be added, changed, or deleted as appropriate. Sales staff must make their case in this meeting for any change in product rollout. Developers must argue for architectural resources. Any company or customer impact is dealt with during the meeting. For example, a change that would directly impact multiple customers will result in an action plan with specific people identified to talk with each customer before the end of the day.

Having the entire company driven by a single agenda out of the MetaScrum meeting has dramatically reduced company communication problems, customer angst, general churn, and confusion. Just as the Scrum meeting has consolidated all decision making for a Sprint, the MetaScrum meeting consolidates all decision making for multiple Sprints. It is a key reason for PatientKeeper success in the marketplace.

5.4.2 Scrum Team Organization

The daily Scrum meeting at PatientKeeper quickly evolved into daily Scrum of Scrum meetings first thing in the morning. All members of the development team are present for 15 minute meetings. Team leaders do most of the reporting, although any contributor may speak to the following:

- 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 tasks will be worked on today? Team members volunteer for tasks. The Scrum Master and the Lead Architect may help bring focus to appropriate tasks.

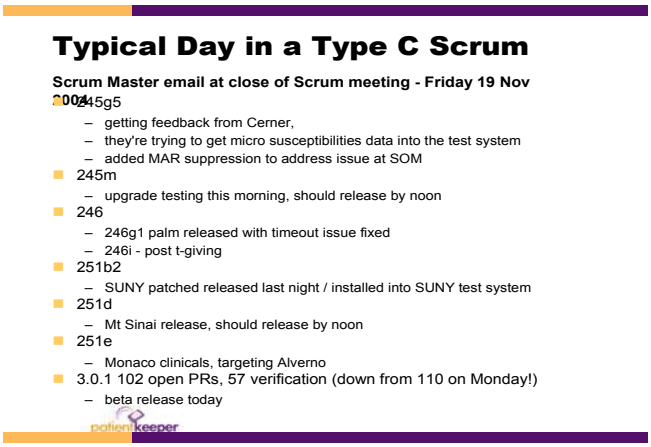

Figure 3: Daily email summary after Scrum of Scrums

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 is done primarily on tasks with difficult design and coding requirements. Many of the developers stay 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.

Every Sprint in the Type C Scrum results in a production code release and all Sprints produce the ultimate demo, i.e. the software goes live and satisfies real customers. The rapid pace of delivery of releases initially created a Quality Assurance (QA) bottleneck. The solution was to assign a small QA team to every release. QA was expanded to four small teams of 2-4 people. This enables them to work with the development team continuously on regression testing and packaging the four top priority releases that are functionality complete while simultaneously doing early testing of developer code on the others. QA is part of the Scrum of Scrums and reports on daily status of ongoing testing.

Thus every Sprint there is a development phase to functionality complete and a packaging phase where the system is regression tested and all critical bugs are eliminated. Developers and QA engineers work closely together from beginning to end of the Sprint. In the packaging phase, developers are focused on eliminating bugs as fast as QA can find them and doing testing when needed. While some view this as a mini-waterfall, it is simply executing the Scrum prime directive – do what makes common sense. Code must be frozen to be regression tested prior to shipment in order to deliver a quality product.

5.4.3 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 [16]. 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 collected daily from developers for tasks. All other data analysis and reporting was automated.

5.4.4 Tools for Data Collection and Reporting

PatientKeeper uses the open source GNATS bug tracking system [14]. Since developers needed to use the bug tracking system daily, there was no additional time overhead for opening the application to enter task data.

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.

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 first 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.

The cumulative amount of work remaining for a release can be obtained 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 when it is touched, the phenomenon of statistical regression towards the mean 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 [13] without advertising that to developers other than showing a very accurate Burndown Chart.

This approach can be generalized to be used for tracking of development tasks within any bug tracking system. Ideally the tracking system integrates with the code versioning system. Tools such as Trac [2] integrate with the Subversion code versioning system [1] and support comprehensive integration of development, error tracking, and code management. These are current candidates for upgrading the GNATS system.

5.5 Type C Scrum Rationale

As noted in our Pattern Languages of Program Design paper [4], “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 [22] 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 [9, 8], 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."[4] When extending Scrum company wide, the entire company self-organizes 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.

5.6 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:

1. Diamond – 320 current open – cumulative work remaining
2. Triangle – 320 daily closed - items closed by QA each day
3. Star – 320 total closed - cumulative closed (on scale at right)
4. Square – 320 current verification - current total in verification (items QA needs to test and close)
5. X – 320 daily open – new tasks opened per day

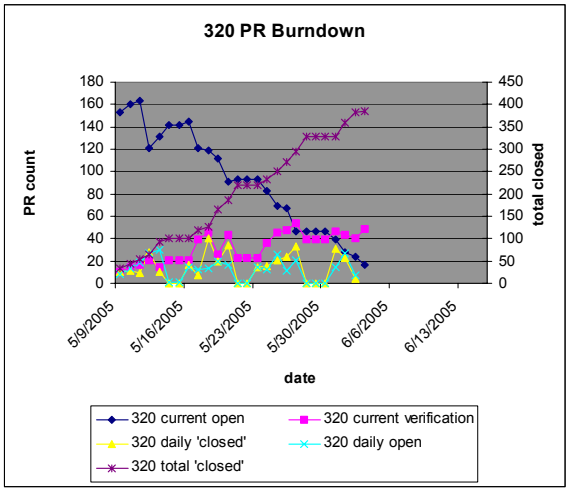

Figure 4: All-In-One Streamlined Burndown Chart showing daily task inflow/outflow and cumulative project churn [15].

The cumulative closed (right scale) is much higher than the starting number of about 160 tasks (left scale). The reason for this is that 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 installs. 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.

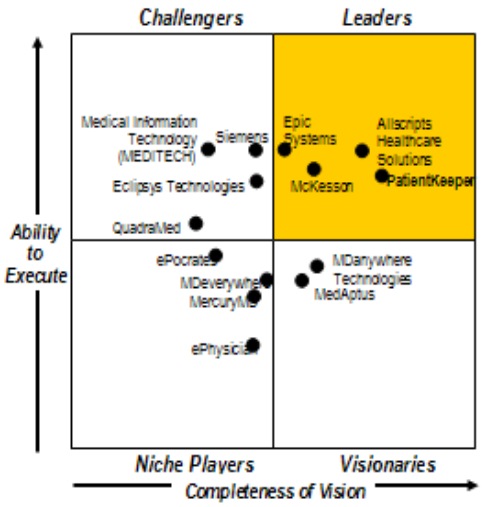

Figure 5: Gartner Group "magic quadrant" for healthcare mobile applications [11].

6. 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 into an organization where Scrum becomes mission critical for the entire enterprise, not just software development.

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. It also drives functionality out into the marketplace at a pace that can overwhelm competitors and achieve industry dominance.

7. References

[1] *Subversion Version Control System*. 2005, Tigris.org: Open Source Software Engineering Tools.

[2] *Trac Integrated SCM and Project Management*. 2005, Edgewall Software Services.

[3] Abdel-Hamid, T.K., *The Slippery Path to Productivity Improvement.* IEEE Software, 1996. **13**(4): p. 43-52.

[4] Beedle, M., et al., *SCRUM: A Pattern Language for Hyperproductive Software Development*, in *Pattern Languages of Program Design*, N. Harrison, Editor. 1999, Addison-Wesley. p. 637-651.

[5] Brooks, F.P., *The Mythical Man Month: Essays on Software Engineering*. 1995: Addison-Wesley.

[6] Cohn, M., *Diagram of Simultaneous Overlapping Sprints*, J. Sutherland, Editor. 2005, Mountain Goat Software.

[7] Goldratt, E.M. and J. Cox, *The goal: a process of ongoing improvement*. 2nd rev. ed. 1994, Great Barrington, MA: North River Press. 351 p.

[8] Holland, J.H., *Emergence: from chaos to order*. 1998, Reading, Mass.: Addison-Wesley. xiii, 258 p.

[9] Holland, J.H., *Hidden order: how adaptation builds complexity*. 1995, Reading, Mass.: Addison-Wesley. xxi, 185.

[10] Jacobson, I., *Object-Oriented Software Engineering: A Use Case Driven Approach*. 1992: Addison-Wesley.

[11] Kleinberg, K. and T. Berg, *Mobile Healthcare: Applications, Vendors and Adoption*, in *Strategic Analysis Report*, R-17-7369, Editor. 2002, Gartner Group. p. 1-44.

[12] MacCormack, A., et al., *Exploring Tradeoffs Between Productivity and Quality in the Selection of Software Development Practices.* IEEE Software, 2003(Sep/Oct): p. 78-85.

[13] McCarthy, W.E., *The REA Accounting Model: A Generalized Framework for Accounting Systems in a Shared Data Environment.* The Accounting Review, 1982. **LVII**(3): p. 554-578.

[14] Osier, J.M., B. Kehoe, and Y. Svendsen, *Keeping Track: Managing Messages with GNATS, The GNU Problem Report Managment System Version 4.0*. 2002, Free Software Foundation.

[15] Salitsky, B., *Scrum Burndown Chart, Release 3.20*. 2005, PatientKeeper, Inc.: Brighton, MA.

[16] Schwaber, K. and M. Beedle, *Agile software development with scrum*. Series in agile software development. 2002, Upper Saddle River, NJ: Prentice Hall. xvi, 158 p.

[17] Sutherland, J., *Agile Can Scale: Inventing and Reinventing SCRUM in Five Companies.* Cutter IT Journal, 2001. **14**(12): p. 5-11.

[18] Sutherland, J., *Agile Development: Lessons Learned from the First Scrum.* Cutter Agile Project Management Advisory Service: Executive Update, 2004. **5**(20): p. 1-4.

[19] Sutherland, J., *Future of Scrum: Pipelining of Sprints in Complex Projects with Details on Scrum Type C Tools and Techniques*. 2005, PatientKeeper, Inc.: Brighton, MA. p. 1-27.

[20] Takeuchi, H. and I. Nonaka, *Hitotsubashi on Knowledge Management*. 2004, Singapore: John Wiley & Sons (Asia).

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

[22] Weinberg, G., *Quality Software Management Vol. 4: Anticipating Change*. 1997: Dorset House.