Future of Scrum: Creating a Scrum Company with a Type C All-At-Once Scrum

Jeff Sutherland, Ph.D.
Scrum Inc., MIT/Cambridge Innovation Center 2011

"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 than previous implementations.

1. Scrum Evolution

Evolution occurs in dynamic response to environmental demands. Now that the Scrum community has over 100,000 [redacted] Scrum Masters and hundreds of thousands,perhaps millions 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.

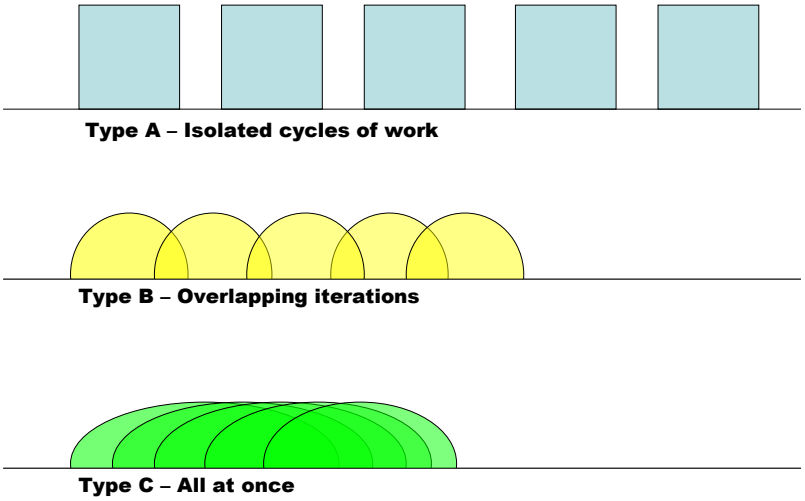

Figure 1: Type A, B, and C strategies for delivering product [11].

After discussing the notion of various types of Scrum with [redacted] Scrum Masters [redacted] 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 Scrum Master. By prioritizing this list which includes personal, team, and company issues, the Scrum Master 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 Scrum Master 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 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 linkage can be very tight between Product Owner and Scrum Master with highly skilled people. For example, at PatientKeeper, the mobile device development team leader is the lead designer and coder on the team, the Scrum Master, 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 Scrum Masters. 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 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, other players move in and can make modifications to improve the timing of the next Sprint.

New Scrum Masters 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 Scrum Master must be able to drive the race car and the Product Owner must be able to direct the Scrum Master 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 Scrum Master 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 Scrum Master 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.

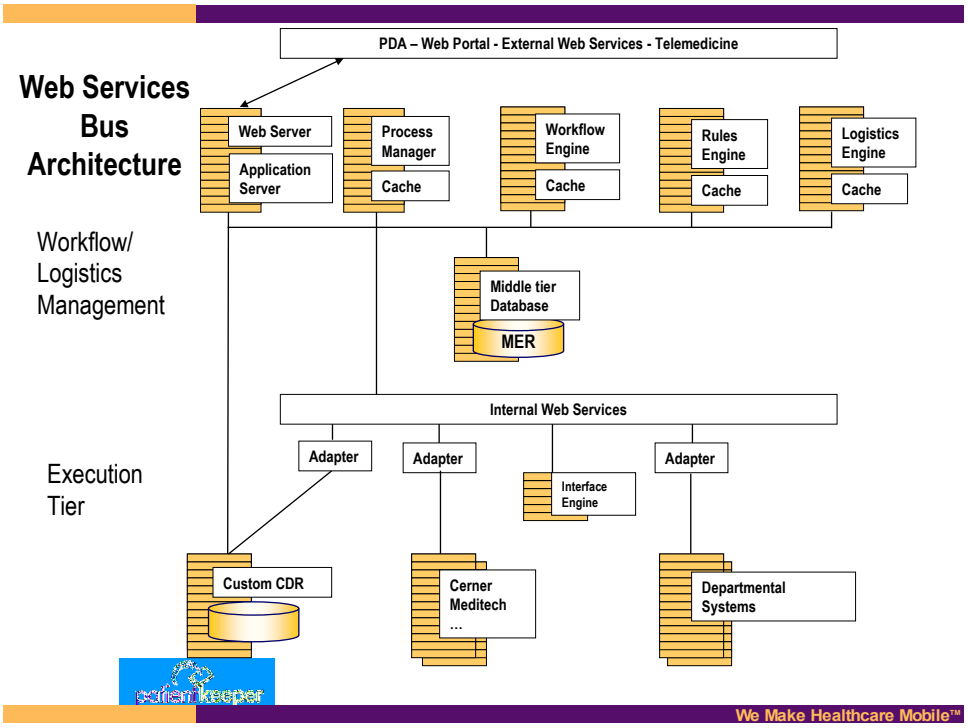

Figure 2: Patientkeeper platform architecture [99].

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. 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. 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 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?

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 3: 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 Scrum Master and the Lead Architect may help bring focus to appropriate tasks.

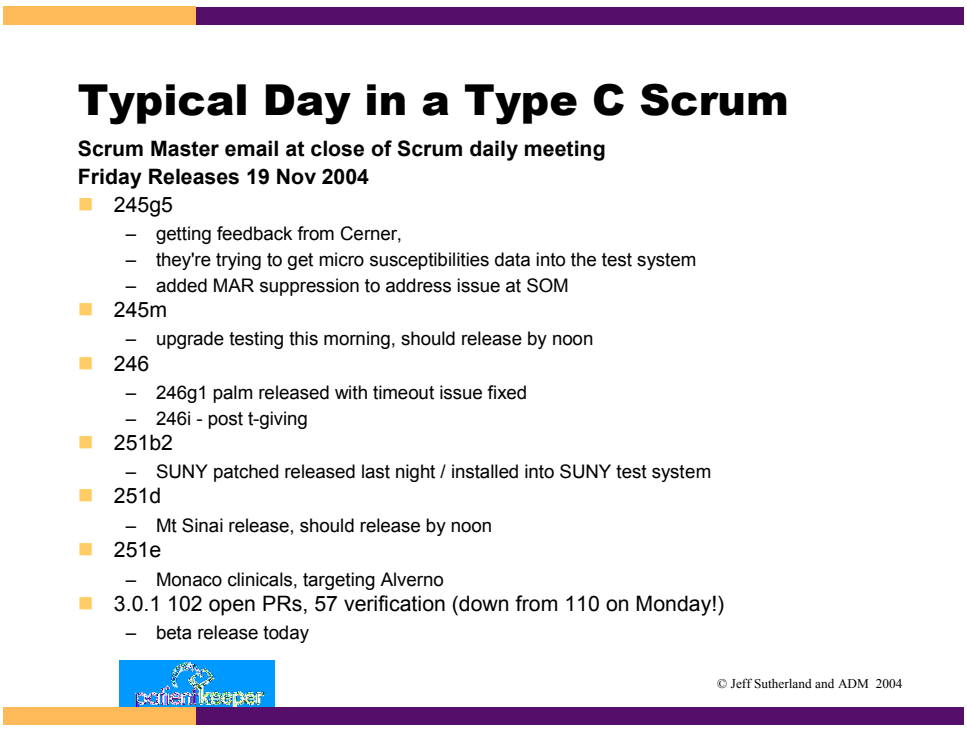

Figure 4: Summary of daily email after Scrum of Scrums meeting shows seven releases in progress simultaneously. All teams work on all releases and all releases result in customer deployment.

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 5: 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 6: 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.

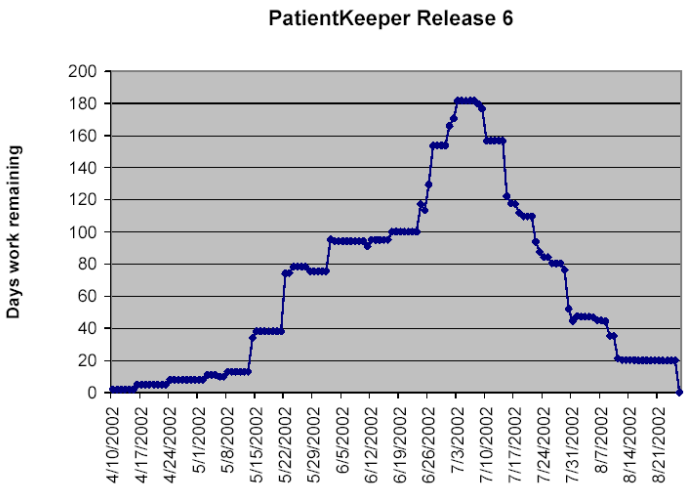

Figure 7: Scrum burndown chart autogenerated by GNATS. Product management was entering placeholders from product backlog until 6/12/2002 when the sprint started. It extended beyond 30 days for reasons described elsewhere. The pain of this Scrum reconvinced everyone that 30 days is the maximum Scrum length.

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.

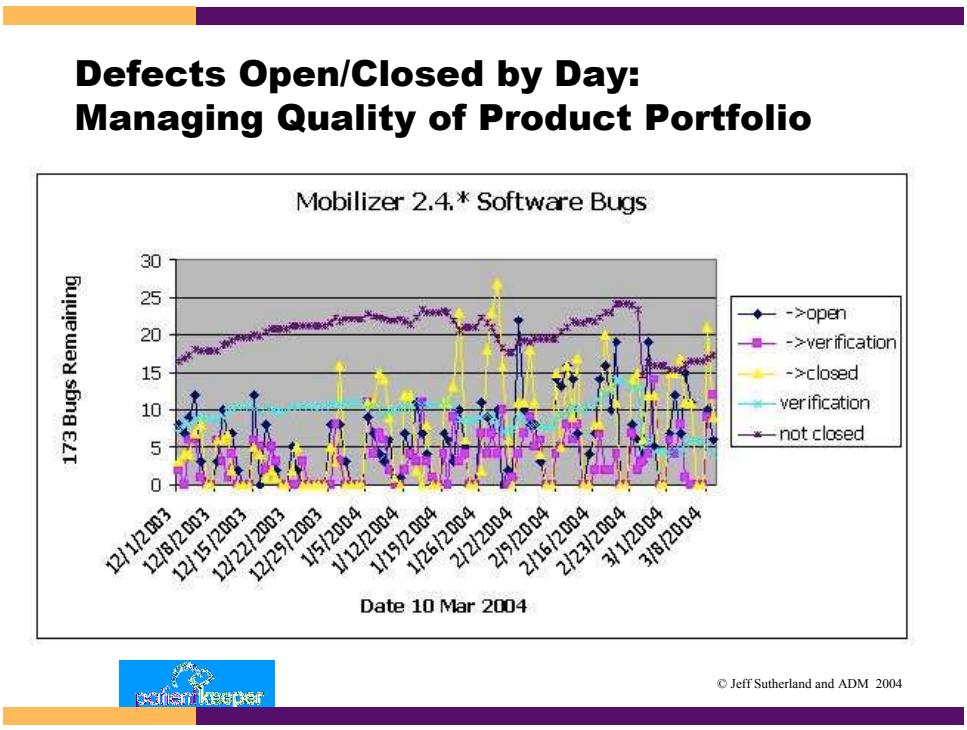

Figure 8: Daily arrival of defects along with cumulative defect count. Company objective is to drive total bug count across a deployed portfolio of releases below 100. This is challenging as there are about 100 hospitals deployed on the 2.4.* releases.

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 Scrum Masters. 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.

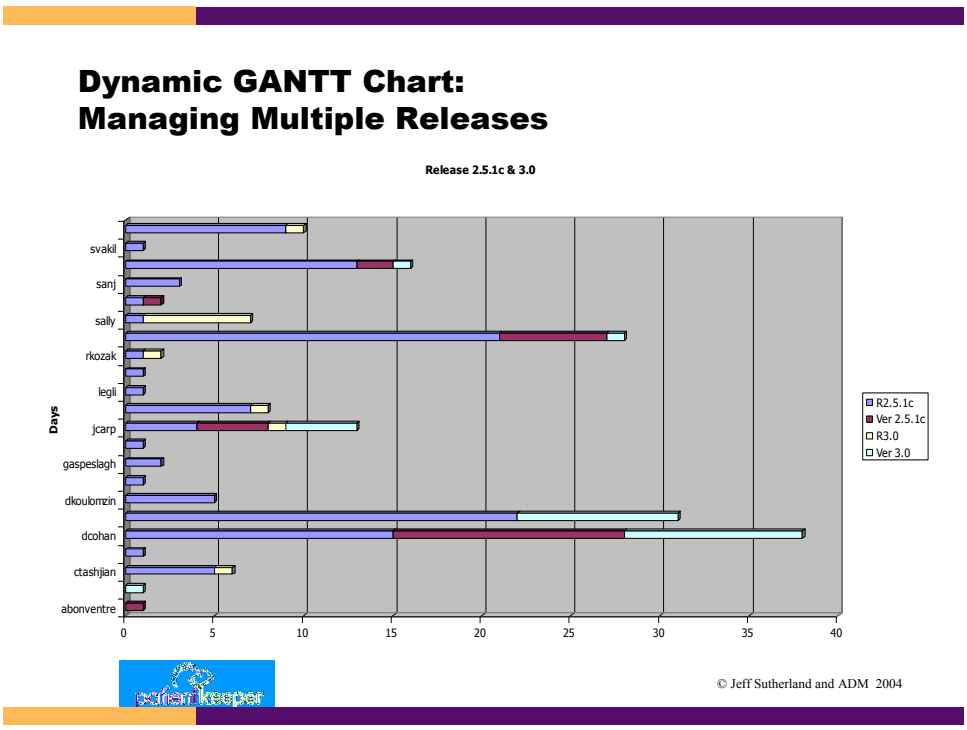

Figure 9: Dynamically generated Gantt chart. End points reflect anticipated release date in days from day of generation.

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

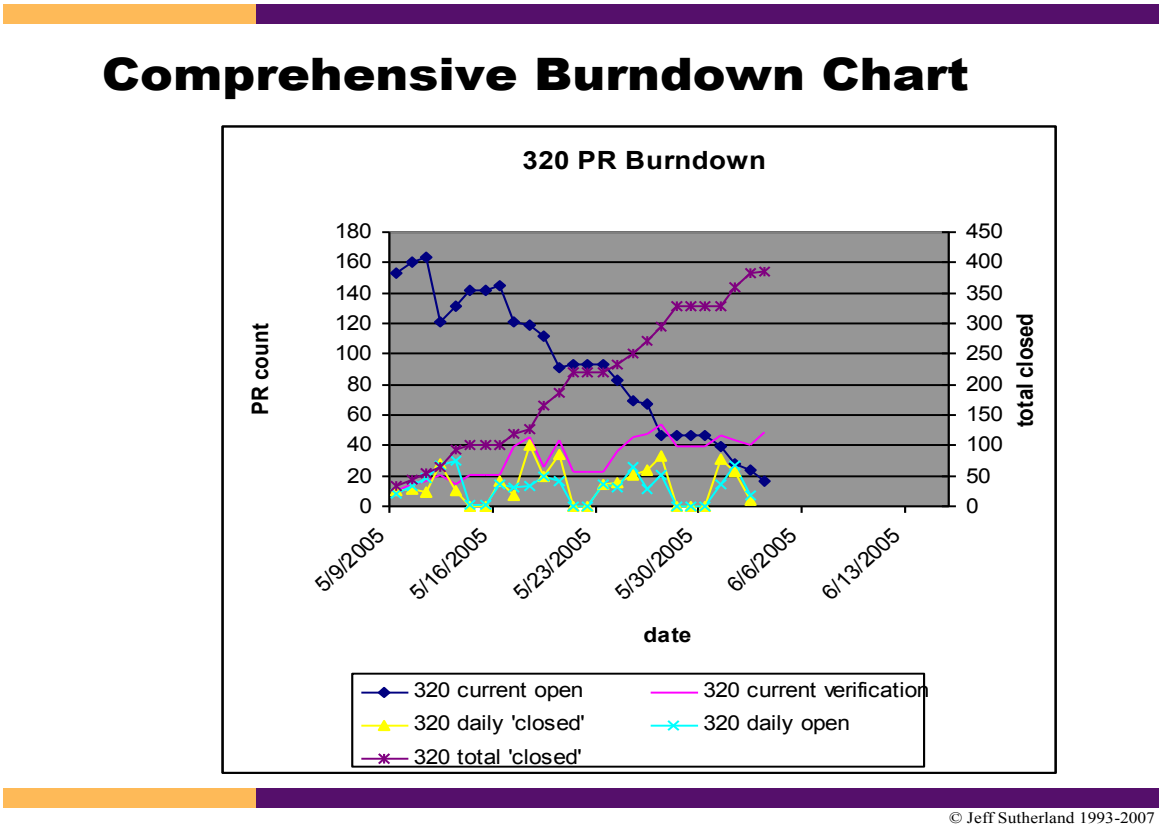

Figure 10: Comprehensive Burndown Chart showing daily task inflow/outflow and cumulative project churn [15].

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.

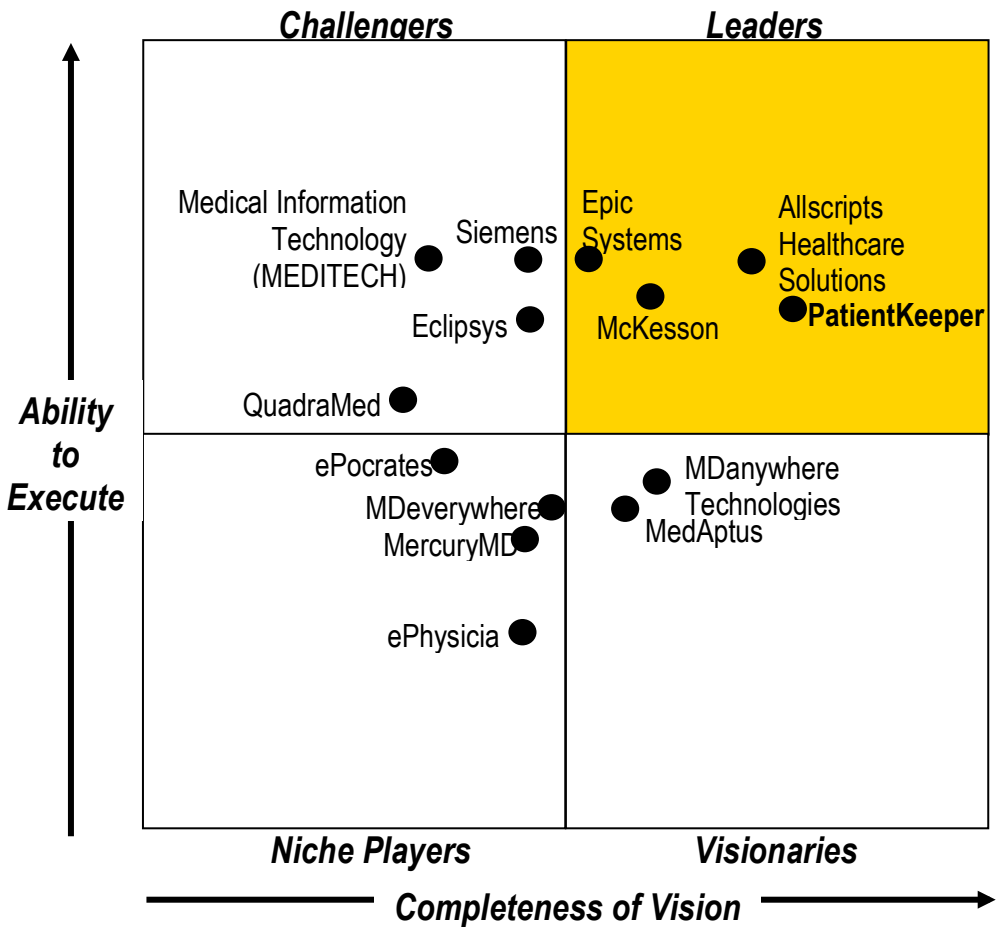

Figure 11: Gartner Group "magic quadrant" for healthcare mobile applications [106].

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.