Adaptive Engineering of Large Software Projects with Distributed/Outsourced Teams

Jeff Sutherland
PatientKeeper, Inc.
jeff.sutherland@computer.org

Anton Viktorov
Starsoft Development Labs
anton.viktorov@starsoftlabs.com

Jack Blount
SirsiDynix
jack@dynix.com

1. Introduction

Scrum is an Agile software development process designed to add energy, focus, clarity, and transparency to project teams developing complex systems. It leverages artificial life research [Langton 1992] by allowing teams to operate close to the edge of chaos to foster rapid system evolution. It capitalizes on robot subsumption architectures [Brooks 1991] by enforcing a simple set of rules that allows rapid self-organization of software teams to produce systems with evolving architectures. A properly implemented Scrum will increase speed of development, align individual and organization objectives, create a culture driven by performance, support shareholder value creation, achieve stable and consistent communication of performance at all levels, and enhance individual development and quality of life.

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

We needed a development process that supported small teams where visualization of design could result immediately in working code. This led to an extensive review of the literature and the real experience from leaders of hundreds of software development projects. Key factors that influenced the introduction of Scrum at Easel Corporation were fundamental problems inherent in software development [DeGrace and Stahl 1990].

- Requirements are not fully understood before the project begins,
- Users know what they want only after they see an initial version of the software,
- Requirements change often during the software construction process,
- And new tools and technologies make implementation strategies unpredictable

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

1.1. "All-at-Once" Development Models

The simplest "All-at-Once" model is a single super-programmer creating and delivering an application from beginning to end. This is the fastest way to deliver a product that has good internal architectural consistency and is the "hacker" model of implementation. For example, in a predecessor to the first Scrum, one individual spent two years writing every line of code for the Matisse object database [Matisse Software 2003] used to drive $10B nuclear reprocessing plants worldwide. At less than 50,000 lines of code, the nuclear engineers said it was the fastest and most reliable database ever benchmarked for nuclear plants.

IBM has shown that a variant of this approach called the Surgical Team is the most productive software development process [Brooks 1995]. The Surgical Team approach has a fatal flaw in that there are at most one or two individuals even in a large company that can execute this model. For example, it took three years for an outstanding team of developers to understand the conceptual elegance of the Matisse object server well enough to maintain it. The single-programmer model does not scale well to large projects.

The next level of "All-at-Once" development is handcuffing two programmers together, as in pair programming in the eXtreme Programming paradigm [Beck 1999]. Here, two developers working at the same terminal deliver a component of the system together. This has been shown to deliver better code (usability, maintainability, flexibility, extendibility) faster than two developers working individually [Wood and Kleb 2003]. The challenge is to achieve a similar productivity effect with more than two people.

Scrum, a scalable, team-based "All-at-Once" model, was motivated by the Japanese approach to team-based new product development combined with simple rules to enhance team self-organization as used in the Brooks subsumption architecture [Brooks 1991]. At Easel we were already using an iterative and incremental approach to building software [Larman 2004]. It was implemented in slices in which an entire piece of fully integrated functionality worked at the end of an iteration. What intrigued us was Takeuchi and Nonaka's description of the team-building process for setting up and managing a Scrum [Takeuchi and Nonaka 1986]. The idea of building a self-empowered team in which a daily global view of the product caused the team to self-organize seemed like the right idea. The approach to managing the team, which had been so successful at Honda, Canon, and Fujitsu, also resonated with the systems thinking approach promoted by Professor Senge at MIT [Senge 1990].

1.2. Hyperproductivity in Scrum

The hyperproductive state achieved in 1993-1994 during the first Scrum was the result of three primary factors. The first was the Scrum process itself, characterized by 15 minute daily meetings where each person answers three questions – what did you accomplish yesterday, what will you do today, and what impediments are getting in your way? This is now part of the definitive Scrum organizational pattern [Beedle, Devos et al. 1999]. Second, the team implemented all XP engineering processes [Beck 1999] including pair programming, continuous builds, and aggressive refactoring. And third, the team systematically stimulated rapid evolution of the software system. Development tasks, originally planned to take days, could often be accomplished in hours using someone else's code as a starting point.

One of the interesting complexity phenomena of the first Scrum was an observed "punctuated equilibrium" effect [Gould 2002]. This occurs in biological evolution when a species is stable for long periods of time and then undergoes a sudden jump in capability. Dennis Hillis simulated this effect on an early super-computer, the Connection Machine.

"The artificial organisms in Hillis's particular world evolved not by steady progress of hill climbing but by the sudden leaps of punctuated equilibrium… with artificial organisms Hillis had the power to examine and analyze the genotype as easily as the realized phenotypes… While the population seemed to be resting during the periods of equilibrium … the underlying genetic makeup was actively evolving. The sudden increase in fitness was no more an instant occurrence than the appearance of a newborn indicates something springing out of nothing; the population seemed to be gestating its next jump. Specifically, the gene pool of the population contained a set of epistatic genes that could not be expressed unless all were present; otherwise the alleles for these genes would be recessive." [Levy 1993]

A fully integrated component design environment leads to unexpected, rapid evolution of a software system with emergent, adaptive properties resembling the process of punctuated equilibrium. Sudden leaps in functionality resulted in earlier than expected delivery of software in the first Scrum.

This aspect of self-organization is now understood as a type of Set-Based Concurrent Engineering (SBCE) practiced at Toyota [Sobek, Ward et al. 1999]. Developers consider sets of possible solutions and gradually narrow the set of possibilities to converge on a final solution. Decisions on how and where to implement a feature in a set of components was delayed until the last possible moment. The most evolved component is selected "just in time" to absorb new functionality, resulting in minimal coding and a more elegant architecture. Thus emergent architecture, a core principle in all Agile processes, is not random evolution. Properly implemented, it is an SBCE technique viewed as a best business practice in some of the world's leading corporations.

2. The SirsiDynix Distributed Scrum

The hyperproductive state achieved by many Scrum teams has increased productivity by an order of magnitude. The question for this paper is whether a large, distributed, outsourced team can achieve the same effect.

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

Best current Scrum practice is for local Scrum teams at all sites to synchronize once a day via a Scrum of Scrums meeting. Here we describe something rarely seen on large, distributed teams. At SirsiDynix, all Scrum teams consist of developers distributed across different sites. Any team member from any site can work on any team task. While some Agile companies operate in this geographically transparent manner on a small scale, SirsiDynix has been successful in using fully integrated Scrum teams with over 50 developers in the U.S., Canada, and Russia. They have created a new implementation of platform and system architecture for a complex Integrated Library System (ILS). An ILS system can best be compared to a vertical market ERP system with a public portal interface used by more than 200 million people. New best practices for distributed Scrum seen on this project consist of (1) daily Scrum meetings of all developers from multiple sites, (2) daily meetings of Product Owner team (3) hourly automated builds from one central repository, (4) no distinction between developers at different sites on the same team, (5) and seamless integration of XP practices like pair programming with Scrum. While similar practices have been implemented on small distributed Scrum teams [Sutherland 2001] this is the first documented project that demonstrates Scrum hyperproductivity for large distributed/outsourced teams building complex enterprise systems.

3. Distributed Team Models

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

Isolated Scrums - Teams are isolated across geographies. In many cases off-shore teams are not cross-functional. In some cases, offshore teams are non-Scrum teams.

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

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

Most outsourced development efforts use a degenerative form of the Isolated Scrums model where outsourced teams are not cross-functional and not Agile. Requirements may be created in the U.S. and developed in Dubai, or development may occur in Germany and quality assurance in India. The authors have experienced cross-cultural communication problems compounded by disparities in work types in many companies around the world where they were directly responsible for development projects. In the worst case, outsourced teams are not using Scrum and their productivity is typical of inhouse waterfall projects further delayed by lag time induced by cross-continent communications.

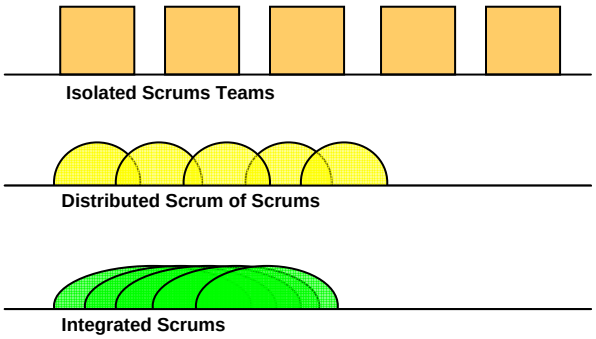

Figure 1: Strategies for distributed Scrum teams [Takeuchi and Nonaka 2004].

The latest thinking in the Project Management Institute Guide to the Project Management Body of Knowledge (PMBOK) models is a degenerative case of isolated non-Scrum teams [Nidiffer and Dolan 2005]. This is a spiral waterfall methodology which layers the Unified Modeling Language (UML) and the Rational Unified Process (RUP) onto teams which are not cross-functional [Zanoni and Audy 2003]. It partitions work across teams, creates teams with silos of expertise, and incorporates a phased approach laden with artifacts that violate the principles of lean development [Poppendieck 2005].

Best practice recommended by the Scrum Alliance is a Distributed Scrum of Scrums model. This model partitions work across cross-functional, isolated Scrum

teams while eliminating most dependencies between teams. Scrum teams are linked by a Scrum-of-Scrums where ScrumMasters (team leaders/project managers) meet regularly across locations. This encourages communication, cooperation, and cross-fertilization.

An Integrated Scrums model has all teams fully distributed and each team has members at multiple locations. While this appears to create communication and coordination burdens, the daily Scrum meetings help to break down cultural barriers and disparities in work styles. On large enterprise implementations, it can organize the project into a single whole with a rapidly evolving global code base. The virtual nature of this approach provides location transparency and creates performance characteristics similar to a small co-located team. The hyperproductive Web team at IDX Systems during 1996-2000 achieved ten times the performance of the industry average for teams of large systems [Sutherland 2001]. The SirsiDynix model outlined in this paper is a good example of best practices for Integrated Scrums. This may be the most productive distributed team ever documented, delivering a large Java enterprise system with more than one million lines of code.

4. SirsiDynix Case Study

4.1. SirsiDynix Background

SirsiDynix provides global technology solutions for libraries to assist people in discovering and using knowledge, resources and other valuable content for their educations, jobs and entertainment. In concert with key industry partners, SirsiDynix supports this strategic role for libraries by offering a comprehensive integrated suite of technology solutions for improving the internal productivity of libraries and enhancing their capabilities for meeting the needs of people and communities. SirsiDynix has approximately 4,000 library and consortia clients, serving more than 200 million people through more than 20,000 library outlets in the Americas, Europe, Africa, the Middle East and Asia-Pacific.

Jack Blount, President and CEO of Dynix and now CTO of the merged SirsiDynix company, negotiated an outsource agreement with StarSoft who staffed the project with more than 20 qualified engineers in less than 60 days. Significant development milestones were completed in just a few weeks and all joint development projects were efficiently tracked and continue to be on schedule.

4.2. StarSoft Background

StarSoft Development Labs, Inc. is a fast-growing software outsourcing service provider in Russia and Eastern Europe. Headquartered in Cambridge, Massachusetts, USA, StarSoft operates development centers in St. Petersburg, Russia and Dnepropetrovsk, Ukraine, employing over 450 professionals. StarSoft has experience handling development efforts varying in size and duration from just several engineers working for a few months to large-scale projects involving dozens of developers and

spanning over several years. A CMM Level 3 company, StarSoft successfully uses Agile development methodologies for the benefits of its clients.

5. Hidden Costs of Outsourcing

The hidden costs of outsourcing can be significant beginning with startup costs. Barthelemy [Barthelemy 2001] surveyed 50 companies and found that 14% of outsourcing operations were failures. In the remainder, costs of transitioning to a new vendor often canceled out most of the company's savings from lower labor costs in other countries. The average time from evaluating outsourcing to beginning of vendor performance was 18 months for projects smaller than the SirsiDynix contract. As a result, the MIT Sloan Management Review counsels readers not to outsource critical IT functions and to spend more time planning. The German Institute for Economic Research analyzed 43,000 German manufacturing firms from 1992-2000 and found that outsourcing services led to poor corporate performance, while outsourcing production helped [Gorzig and Stephan 2002]. While this is a manufacturing study rather than software development, it suggests that outsourcing core development may provide gains not seen otherwise.

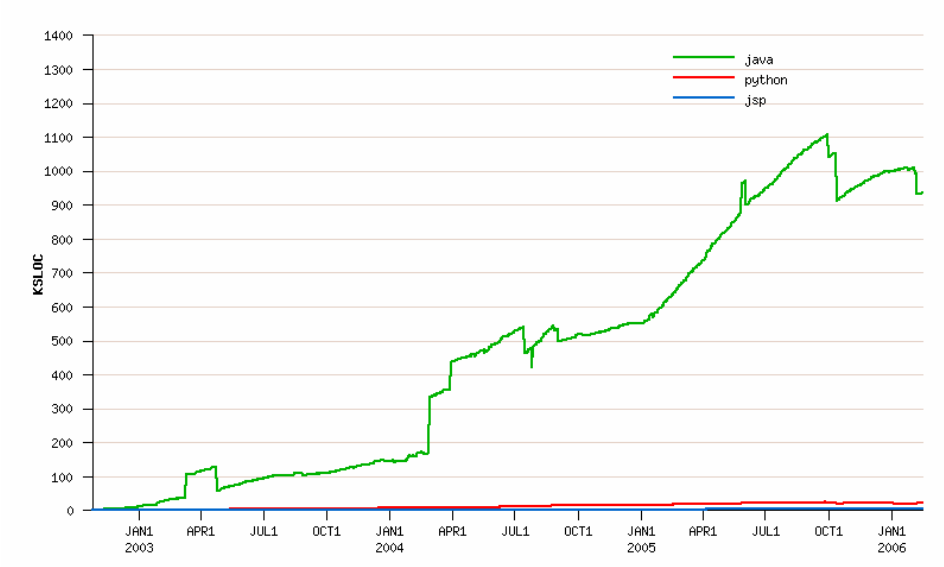

Figure 2 – SirsiDynix lines of new Java code in thousands from 2000-2006.

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

SirsiDynix sidestepped many of the hidden costs, directly outsourced primary production and used Integrated Scrums to control the risk. The goals of both increasing

output per team member and increasing overall output by increasing team size were achieved. Production velocity more than doubled when they increased the size of the 30 person North American development team and added 26 people from StarSoft on 1 December 2005.

6. Intent of the Integrated Scrums Model

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

7. Context

Software complexity and demands for increased functionality are exponentially increasing in all industries. When the lead author of this paper flew F-4 aircraft in combat in 1967, 8% of pilot functions were supported by software. In 1982, the F16 software support was 45%, and by 2000, the F22 was augmented 80% of pilot capabilities with software [Nidiffer and Dolan 2005]. Demands for ease of use, scalability, reliability, and maintainability increase with complexity.

SirsiDynix was confronted with the requirement to completely re-implement a legacy library system with over 12,500 installed sites across the globe. The large number of developers required over many years in the midst of a changing business environment threatened to obsolete many feature requirements in the middle of the project. To complicate matters further, the library software industry was in a consolidating phase. Dynix started the project in 2002 and merged with Sirsi in 2005 to form SirsiDynix.

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

The combination of high risk, large scale, changing market requirements, merger and acquisition business factors, and the SirsiDynix experience with Scrum combined with StarSoft success with XP led them to choose an Integrated Scrums implementation. Jack Blount's past experience with Agile development projects at US Data Authority, TeleComputing and JD Edwards where he had used Isolated Scrums and Distributed Scrum of Scrums models was a key factor in his decision to structure the project as Integrated Scrums.

8. Forces

8.1. Complexity Drivers

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

SSCI complexity drivers are described as [Nidiffer and Dolan 2005]:

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

SirsiDynix faced all of these issues. Legacy products were difficult to sell to new customers. They needed a new product with complete functionality for the library enterprise based on today's technologies that was highly scalable, easily expandable, and used the latest computer and library standards,

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

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

8.2. Top Issues in Distributed Development

The SSCI has carefully researched top issues in distributed development [Nidiffer and Dolan 2005], all of which had to be handled by SirsiDynix and StarSoft.

- Strategic: Difficult leveraging available resources, best practices are often deemed proprietary, are time consuming and difficult to maintain.

- Project and process management: Difficulty synchronizing work between distributed sites.
- Communication: Lack of effective communication mechanisms.
- Cultural: Conflicting behaviors, processes, and technologies.
- Technical: Incompatible data formats, schemas, and standards.
- Security: Ensuring electronic transmission confidentiality and privacy.

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

9. Solution: Integrated Scrums

There are three roles in a Scrum: the Product Owner, the ScrumMaster, and the Team. SirsiDynix used these roles. Scrum itself solves the strategic distribution problem of building a high velocity, real-time reporting organization with an open source process that is easy to implement and low-overhead to maintain [Sutherland 2005].

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

9.1. Team Formation

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

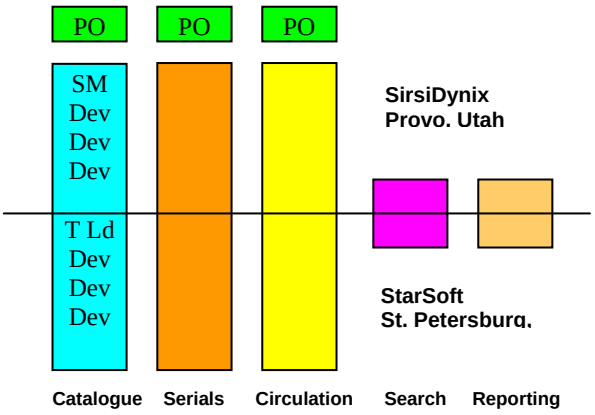

Figure 3 – Scrum teams split across sites. PO=Product Owner, SM=ScrumMaster, TLd=Technical Lead.

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

9.2. Scrum Meetings

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

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

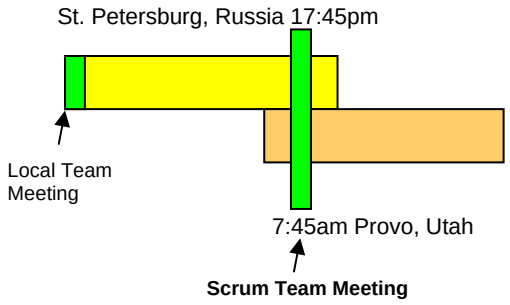

Figure 5 – Scrum Team meetings

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

ScrumMasters are all in Provo, Utah or Waterloo, Canada, and meet in a Scrum of Scrums every Monday morning. Here work is coordinated across teams. Architects are directly allocated to production Scrum teams and all located in Utah. An Architecture group also meets on Monday after the Scrum of Scrums meeting and controls the direction of the project architecture through the Scrum meetings. A Product Owner resident in Utah is assigned to each Scrum team. A chief Product Owner meets regularly with all Product Owners to assure coordination of requirements.

SirsiDynix achieved strong central control of teams distributed across geographies by centrally locating ScrumMasters, Product Owners, and Architects. This enabled them to get consistent performance across all distributed teams.

9.3. Sprints

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

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

Code is feature complete and demoed at the end of each Sprint. If it meets the Product Owner's functional requirement, it is considered done. It is not deliverable code and SirsiDynix wants to strengthen its definition of "done" to include testing. Failure to do this allows work in progress to cross Sprint boundaries, introducing wait times and greater risk into the project.

9.4. Product Specifications

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

Story for Simple Renewals Use Case - Patron brings item to staff to be renewed.

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

Assumptions:

- Item being renewed is currently checked out to the active patron
- No requests or reservations outstanding
- Item was not overdue
- Item does not have a problem status (lost, cr, etc)
- No renew maximums have been reached
- No block/circ maximums have been reached

- Patron's subscriptions are active and not within renewal period
- No renewal charges apply
- No recalls apply
- Renewal is from Check Out (not Check In)
- Staff User has renewal privileges

Verification (How to verify completion):

- Launch Check Out
- Retrieve a patron who has an item already checked out but not yet overdue
- Enter barcode for checked out item into barcode entry area (as if it is being checked out), and press <cr>.
- System calculates new due date according to circ rules and agency parameters.
- The renewal count is incremented (Staff renewal with item)
- If user views "Circulation Item Details", the appropriate Renewals information should be updated (renewals used/remaining)
- Cursor focus returns to barcode entry area, ready to receive next scan (if previous barcode is still displayed, it should be automatically replaced by whatever is entered next)
- A check of the item and patron circulation statistics screens show a new row for the renewal with the renewal date/time and the new due date.

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

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

9.5. Testing

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

The test first approach is encouraged although not mandated. Tests are written simultaneously with code most of the time. GUIs are not unit tested. Manual testing is not currently complete for the Sprint Demo leading to a lot of open work in progress.

Component	Test Cases	Tested
Acquisitions	529	384
Binding	802	646
Cataloging	3101	1115

Circulation	3570	1089
Common	0	0
ERM	0	0
Pac Searching	1056	167
Serials	2735	1714
Sub Total	11793	5115

Figure 4 – Test Cases Created vs. Tested

During the Sprint, the Product Owner tests features that are in the Sprint backlog. Testers receive a stable Sprint build only after the Sprint demo. The reason for this is a low tester/developer ratio.

There are 30 team members in North America and 26 team members in St. Petersburg on this project. The St. Petersburg team has one project leader, 3 technical team leaders, 18 developers, 1 test lead, and 3 testers. This low tester/developer ratio and makes it impossible to have a fully tested package of code at the end of the Sprints. Fixing this problem could accelerate production in the future.

9.6. Configuration Management

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

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

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

9.7. Pair Programming, Refactoring, and other XP practices

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

9.8. Measuring Progress

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

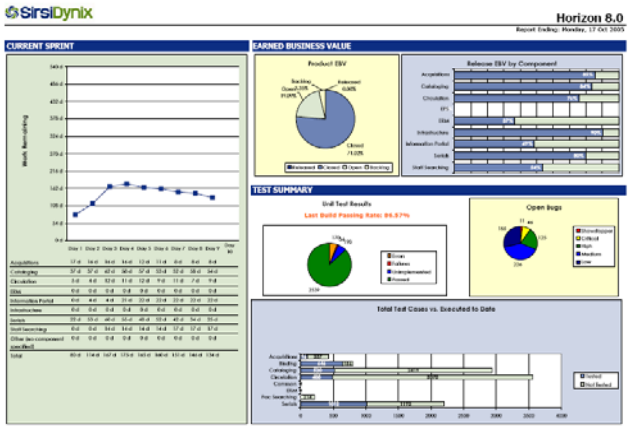

Figure 6 – SirsiDynix Horizon 8.0 Project Dashboard

Data from Jira can be downloaded into Excel to create any requested data analysis. High velocity complex projects need an automated tool to track status across teams and geographies. The best tools support bug tracking and status of development tasks in one system to avoid extra work on data entry by developers. Such tools should track tasks completed by developers and work remaining. They provide more detailed and useful data than time sheets, which should be avoided. Time sheets are extra overhead that do not provide useful information on the state of the project, and are de-motivating to developers.

Other companies like PatientKeeper [Sutherland 2005] have found tools that incorporate both development tasks and defects that can be packaged into a Sprint Backlog are highly useful for complex development projects. Thousands of tasks and dozens of Sprints can be easily maintained and reviewed simultaneously with the right tool.

10. Integrated Scrums Model Resulting Context

Collaboration of SirsiDynix and StarSoft turned the Horizon 8.0 project into one of the most productive Scrum projects ever documented. For example, data is provide in the table below on a project that was done initially with a waterfall team and then re-implemented with a Scrum team [Cohn 2004]. The waterfall team took 9 months

with 60 people and generated 54000 lines of code. It was re-implemented by a Scrum team of 4.5 people in 12 months. The resulting 50,803 lines of code had more functionality and higher quality.

	SCRUM	Waterfall	SirsiDynix
Person Months	54	540	827
Lines of Java	50,803	54000	671,688
Function Points	959	900	12673
FP per dev/month	17.8	2.0	15.3
FP per dev/month (industry average)	12.5	12.5	3

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

Capers Jones of Software Productivity Research has published extensive tables on average number of function points per lines of code for all major languages [Jones 1996]. Since the average lines of code per function point for Java is 53, we can estimate the number of function points in the Scrum application. The waterfall implementation is known to have fewer function points.

Distributed team working on Horizon 8.0 generated 671,688 lines of code in 14.5 months with 56 people. During this period they radically refactored the code on two occasions and reduced the code based by 275,000. They have not been penalized for radical refactoring as that is rarely done in large waterfall projects in the database from which Capers derived his numbers.

Jones has also shown from his database of tens of thousands of projects that industry average productivity is 12.5 function points per developer/month for a project of 900 function points and that this drops to 3 for a project with 13000 function points [Jones 2000].

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

11. Conclusions

It is extremely easy to integrate Scrum with XP practices even on large distributed teams. This can improve productivity, reduce project risk, and enhance software quality.

What is new in this paper is that single teams with members distributed across sites can enhance code ownership and improve autonomy essential to team self-organization. One Scrum meeting a day was necessary which included all team members across geographies. Written communication prior to joint meetings was needed to improve communication and reduce misunderstandings due to cultural and distance barriers. Project leaders in Provo, Utah, and St. Petersburg had a remarkable common view of the project because of the transparency and frequency of communications.

Automated communication of Product and Sprint backlogs throughout the organization combined with upward reporting of Scrum status to management can tightly align a global organization.

The issues of Product Backlog being "ready" for implementation in a Sprint and working software being "done" at the end of a Sprint are key areas where even the best teams need improvement.

References

Barthelemy, J., 2001, "The Hidden Costs of Outsourcing," *MITSloan Management Review* **42**(3): 60-69.

Beck, K., 1999, *Extreme Programming Explained: Embrace Change*, Addison-Wesley (Boston).

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

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

Brooks, R. A., 1991, "Intelligence without representation," *Artificial Intelligence* **47**: 139-159.

Cohn, M., 2004, *User Stories Applied: For Agile Software Development*, Addison-Wesley.

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

Gorzig, B. and A. Stephan, 2002, *Outsourcing and Firm-level Performance*.

Gould, S. J., 2002, *The structure of evolutionary theory*, Belknap Press of Harvard University Press (Cambridge, Mass.).

Jones, C., 1996, *Programming Languages Table, Release 8.2*, (Burlington, MA).

Jones, C., 2000, *Software assessments, benchmarks, and best practices / Capers Jones*, Addison Wesley (Boston, Mass.).

Langton, C. G., 1992, *Life at the Edge of Chaos*, Artificial Life II, SFI Studies in the Sciences of Complexity, Held Feb 1990 in Sante Fe, NM, Addison-Wesley.

Larman, C., 2004, *Agile & Iterative Development: A Manager's Guide*, Addison-Wesley (Boston).

Levy, S., 1993, *Artificial Life: A Report from the Frontier Where Computers Meet Biology*, Vintage, Reprint edition (New York).

Matisse Software, 2003, *The Emergence of the Object-SQL Database*, (Mountain View, CA).

Nidiffer, K. E. and D. Dolan, 2005, "Evolving Distributed Project Management," *IEEE Software* **22**(5): 63-72.

Poppendieck, M., 2005, *A History of Lean: From Manufacturing to Software Development*, JAOO Conference, Aarhus, Denmark, EOS.

Senge, P. M., 1990, *The Fifth Discipline: the Art and Practice of the Learning Organization*, Currency (New York).

Sobek, D. K. I., A. C. Ward, et al., 1999, "Toyota's Principles of Set-Based Concurrent Engineering," *Sloan Management Review* **40**(2): 67-83.

StandishGroup, (2003), "2003 Chaos Chronicles." from http://www.standishgroup.com/press/article.php?id=2.

Sutherland, J., 2001, "Agile Can Scale: Inventing and Reinventing Scrum in Five Companies," *Cutter IT Journal* **14**(12): 5-11.

Sutherland, J., 2005, *Future of Scrum: Parallel Pipelining of Sprints in Complex Projects with Details on Scrum Type C Tools and Techniques*, (Brighton, MA).

Sutherland, J., 2005, *Scrum Evolution: Type A, B, and C Sprints*, Agile 2005 Conference, Denver, CO.

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

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

Wood, W. A. and W. L. Kleb, 2003, "Exploring XP for Scientific Research," *IEEE Software* **20**(3): 30-36.

Zanoni, R. and J. L. N. Audy, 2003, *Projected Management Model for Physically Distributed Software Development Environment*, HICSS'03, Hawaii, IEEE.