Rally Software Development Corporation and
Ken Schwaber-Scrum Alliance

Whitepaper

A CIO's Playbook for Adopting the Scrum Method of Achieving Software Agility

With
Dean Leffingwell
and
Hubert Smits

080805

www.rallydev.com
www.controlchaos.com
www.scrumalliance.org

v 303 565 2800
f 303 226 1179
www.rallydev.com

Table of Contents

Introduction

The pressures of a truly global economy cause today's business to increasingly rely on their ability to produce *software* as a key competitive advantage. Whether it be software for managing manufacturing and customer delivery processes or software improving the efficiency of day to day activities, software touches virtually every facet of today's businesses.

And yet for many organizations, software development practices remain as they were in the 1980s. Reliance on prescriptive, plan-based, waterfall methods is common despite mountains of evidence that these practices often fail to achieve real value delivery in a timely fashion, and so hamper our company's responsiveness to fast-changing customer requirements and market conditions. And it's not getting easier.

Today's IT organizations must also effectively coordinate globally distributed software development teams while re-factoring legacy applications into more flexible, service oriented architectures. Clearly, we need a new approach for managing and developing software to remain competitive.

Managing distributed organizations and effectively migrating to service oriented architectures demand a new software development approach

To address these challenges, a number of more agile and adaptive software development techniques are being adopted which allow organizations to deliver higher quality software more quickly. *Scrum* is one such proven method that has seen widespread adoption in many software organizations. This whitepaper describes how a CIO or other executive manager can implement Scrum on an organization-wide basis, including scaling across larger applications and teams of teams – the challenges he or she will face as well as the rewards – and provides a *playbook* for adopting Scrum in enterprises where software, and lots of it, is the key to competitive success in the marketplace.

This is a "playbook" of ideas about implementing Scrum within an enterprise. It is called a playbook rather than a manual because each organization is unique. Scrum's implementation within one enterprise will be significantly different from its implementation in another. The types of impediments, things that need changing, the difficulty of change, and the people who will be doing the changing are different, so the timetables, the priorities, and the effort will be different as well.

Overview of Scrum and Software Agility

On the surface, Scrum is a very simple process: a software management technique that has a relatively small set of interrelated practices and rules, is not overly prescriptive, can be learned quickly and is able to produce productivity gains almost immediately.

Scrum naturally focuses an entire organization on building successful products. It delivers useful features at regular intervals as requirements, architecture, and design emerge, even when using unstable technologies. You can implement Scrum at the beginning of a project or in the middle of a project, and Scrum has saved many development efforts that were in trouble.

Scrum works because it optimizes the development environment, reduces organizational overhead, and closely synchronizes market requirements with early feature delivery. Based in modern process control theory, Scrum produces the best possible software given the available resources, acceptable quality levels, and required release dates.

At its core, Scrum is an iterative, incremental process for developing any product or managing any work that produces a potentially shippable set of functionality at the end of each iteration. Scrum's attributes are:

- Scrum is an agile process to manage and control development work.
- Scrum is a wrapper for existing engineering practices.
- Scrum is a team-based approach to developing systems when requirements are changing rapidly.
- Scrum controls the chaos of conflicting interests and needs.
- Scrum improves communication and maximizes cooperation.
- Scrum detects and removes anything that gets in the way of developing and delivering products.
- Scrum is a way to maximize productivity.
- Scrum scales from single projects to entire organizations, and has managed development for multiple interrelated products and projects with over a thousand team members.
- Scrum is a way for everyone to feel good about their job, their contributions, and know they have done the very best they possibly could.

While describing Scrum practices in detail is outside the scope of this whitepaper (see Schwaber 2004 and Schwaber 2002), the method is characterized by the production of a *Product Backlog* where requested features are organized by their priority (Figure 1). A *Product Owner* is responsible for approving changes to the product backlog. Implementation occurs in roughly 30 day iterations called *Sprints* which focus on the top priorities in the Product Backlog. The goal of each Sprint is to deliver a potentially shippable product increment. During the Sprint, checkpoints are observed in a daily "Scrum" meeting which communicates the progress and activities within the team and shares issues that may be "blocking" progress for an individual or the team. This allows the ScrumMaster to determine progress against the Sprint commitments and advise on midcourse corrections to assure successful completion of the Sprint.

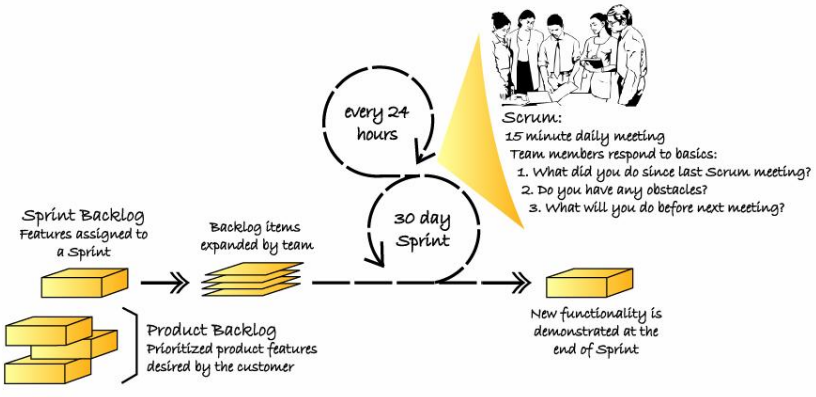

Figure 1: An Empirical Process Model that Characterizes Scrum

Scrum Principles

While those are some of the mechanics of Scrum, more importantly, the CIO should understand that Scrum is guided by a few key principles:

- The belief that effective software development is best implemented via an *empirical* rather than *planned* process;
- The belief that, once organizational impediments are removed, a self organizing and self managing team will naturally deliver better software than would otherwise be the case;
- The premise that you can deliver the most valuable software within a prescribed time and budget, and yet you *cannot* definitively predict the exact functionality of what a team will deliver.

Scrum's assertion is that recognizing these key principles frees an organization from many of the constraints that prevent effective software development. However, CIOs must also recognize that these key principles imply potentially significant *change* to the organization that chooses to adopt them. Since these principles form the underlying basis of Scrum, each merits some additional discussion.

Adopting an Empirical vs. Planned Process

Scrum believes that most systems development today has an incorrect philosophical basis, that is, through more and better planning we can achieve more predictable, higher quality results. Scrum recognizes that the applications development process is an unpredictable and extraordinary complicated process (think hundreds of thousands of manually created lines of code) whose value can only be measured empirically. After all, the application under development has likely not been developed by any team anywhere, ever, much less by your team in your context, so cookbook, step-by-step planning approaches cannot effectively address the inherent unpredictability.

Scrum defines the systems development process as a loose set of activities that combines known, workable tools and techniques with an empowered team that is tightly coupled to the Customer/Product Owner. Since many of these activities are loose, controls are applied – such as constant inspection and demonstration –to manage the risk and provide real time, empirical evidence of the state of the project at every point in time.

The Scrum tradeoff is simple:

Know where you are every day with Scrum

- or -

Think you know where you are on your well-formed plan

and discover that you are very wrong, very much later

Eliminate the Impediments So the Team Can Do its Job

Over the years, a company's organizational processes and software development practices typically gain weight until building software is often quite a difficult endeavor. When Scrum is implemented, these "organizational impediments" to effective software delivery become quite obvious, for they get in the way of the team's ability to deliver on the rapid iterative, incremental nature of Scrum. Removing or changing these processes and practices may show that a major change project must be initiated, driven and monitored by the CIO or executive champion (more on this topic later).

Moreover, in Scrum, *the team is the thing*. After all, they are the ones who actually design, develop and deliver the application, so optimizing their performance by eliminating obstacles optimizes the business's performance in delivering value to its users. Management does their job when they eliminate impediments. The Team does its job when it meets its commitments as described in each Sprint's Backlog.

In other words, in Scrum, the team is both *empowered and accountable* to deliver the goods. The team does their job when they self-organize, self-manage and self-achieve the objectives of the Sprint. For many organizations, this turns things upside down. The hierarchical-technical-management-directive approach is essentially eliminated with Scrum. The Product Owner now sets the objectives and priorities, the team figures out how to achieve them, and no one need tell them how to do that along the way.

Better, Though Less Predictable Outcomes vs. False Confidence

Scrum starts with the premise that creating software is a complicated business operating in a highly-fluid and technical environment, and that no one can reliably predict or definitively plan exactly what a team will deliver, when they will deliver it, and what the quality and cost will be. Instead, Scrum understands that teams can estimate these items, communicate the estimates, negotiate a near term plan according to various risks and then adjust as they proceed. The agreement is that the team *will deliver the best possible software given the circumstances*, and that following any cookbook approach won't improve the definition of "best," and will only hinder the team's responsiveness to the real-world complexity and unpredictability that exists.

Historically, ignoring this philosophy creates a number of organizational problems:

- Management actually believes that it can predict the cost, delivery schedule, and functionality that will be delivered, and plans accordingly.
- Developers and project managers are forced to live a lie: they pretend they can plan, predict and deliver. They build one way, but must pretend they build another way. In the end, they are essentially without controls.
- By the time the system is delivered, it is often irrelevant or requires significant change. A key cause is that high iteration costs limit our visibility into the usefulness of what the team is actually developing, until it is too late.

Recognizing these realities is not without its challenges – for example, what manager wants to tell their executive they don't know exactly *what* the team will deliver on the given date? But the benefits of this approach are that organizations are truly empowered: the business is finally free to produce better outcomes for its end users and will now do so more quickly, clearly creating competitive advantage for the business.

Scrum and Software Agility

Scrum has been in use since the mid 1990s and has now been applied to thousands of projects worldwide. In addition to Scrum, several new iterative methodologies have also received attention during this period. Like Scrum, each had a combination of old ideas and new ideas, but they *all* emphasized:

- Close collaboration between the development team and business experts;
- Face-to-face communication (as more efficient than written documentation);
- Frequent delivery of new deployable business value software;
- Tight, self-organizing teams; and
- Ways to craft the code and the team to allow for continuous adaptation to changing requirements.

In 2001, various originators and practitioners of these methodologies, including Scrum leaders, met to understand what it was they had in common. They picked the word "agile" for an umbrella term and crafted the “Manifesto for Agile Software Development” (Appendix B), its most important aspect being a statement of shared values:

> *We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

> *That is, while there is value in the items on the right, we value the items on the left more.*

The Manifesto struck a chord and it led to the start of thousands of new agile projects. The results and experiences of these projects further enhanced the techniques applied by the multiple forms of agile practices. As with any human endeavor, some succeeded and some failed. But what was most striking about the successes was how much both the business people and the technical people loved their project. This was the way they wanted software development done – and the customers and end users agreed. Successful projects spawned more enthusiasts and like a successful Sprint, the virtuous agile cycle continues today.

Preparing for Scrum

Once CIOs have a basic understanding of the business and cultural benefits of Scrum and agility, they often want to take the next step and see how this development method can improve their organization.

During its first fifteen years of life, most Scrum implementations have been driven bottom-up. In other words, a project team would try Scrum and the results would be impressive. Another team would try it, and pretty soon Scrum projects appear throughout the organization. More recently, however, many organizations want to implement Scrum top-down as part of a directive to speed the company's responsiveness and to improve productivity.

Since Scrum is all about team empowerment and "letting the team decide," a top down implementation requires thoughtful consideration and preparation, as we will describe in this section.

"Scrumming" both the Software Process and the Organization

Many organizations have tolerated inefficiencies and impediments for years; Scrum quickly identifies these and requires their resolution. Fortunately, the increased productivity and value derived from Scrum projects makes the effort worthwhile, but it is still an effort.

To implement Scrum, an organization has to take on two pieces of work. Firstly, projects where development teams are taught to build software using Scrum; and, secondly, removing the impediments to optimized creation and delivery of software that the Scrum teams encounter. The first work improves software delivery; the second remedies impediments to ROI and productivity identified in the first.

Both pieces of work are challenging and require hard work above and beyond the actual development of software; a full Scrum implementation may take up to two years. No matter the intensity or commitment by management, this timetable cannot be rushed because the core of the project is organizational change.

Scrum's daily and monthly inspection and adaptation cycles make everything visible - the code, the process, and the company's impediments. Projects using Scrum regularly identify impediments that must be recorded, evaluated, prioritized, and acted upon.

The speed of Scrum implementation is directly related to the:

- Degree of change required within the organization;
- Urgency within the organization to improve its software development and delivery process;
- Effectiveness of leadership within the organization.

The CIO's Role as *Organizational ScrumMaster* for Continuous Improvement

In Scrum, the *ScrumMaster* is responsible for making sure a Scrum team lives by the values and practices of Scrum. The ScrumMaster protects the team by making sure they do not over-commit themselves to what they can achieve during a Sprint and the ScrumMaster continuously removes impediments that prevent the team from successfully delivering the Sprint results.

The CIO is the ScrumMaster for Organizational Change

At the organizational-impediment level, this job falls to the CIO or other executive sponsor, whose job it is to work outside the team and eliminate the organizational barriers that may prevent the success of an agile development model.

The job of the *Organizational ScrumMaster* is to notice, identify, and work within the organization to cause change that removes impediments. That is, the CIO as Organizational ScrumMaster is primarily a change agent, and the list of impediments is their Product Backlog. The CIO's Scrum sponsor – acting as "Product Owner" for these impediments – sets the priorities of these items. This Product Backlog of impediments is worked on by the organization through teams using the Scrum process, with deliverables being impediment removal. This organizational change backlog starts during the pilot projects and continues as long as needed changes are identified during the inspect-and-adapt cycle of Scrum.

The Organizational ScrumMaster periodically meets with all of the ScrumMasters, product owner and sponsor to further develop the Organizational Change Product Backlog. Teams are formed that drive changes to the organization within a Sprint. At the Sprint Review, the change is reviewed as well as the metric that can be used to monitor progress in implementing the change. In this way, the CIO engages in a process of continued organizational improvement all aimed specifically at increasing the productivity and quality of the software development teams.

Caution: Change is Hard Work

Change is hard work and there is no way around the hard work. Organizations implementing Scrum sometimes misidentify the hard work as someone's fault, something that can be made to go away if the group at fault would just "clean up their act". This type of organizational blame can kill a Scrum implementation, and with it the organization's ability to build better software. When something is painful, when something goes wrong, recognize this is just part of the change that is occurring; it is an opportunity for everyone to get together to figure out how to solve the problem, together.

Scrum cannot be planned for and implemented with checklists, procedures, and forms. Scrum is just a simple framework that will identify everything in an organization that gets in the way of optimally building software. The work to manage and remove these impediments represents the difficult part of implementing Scrum, and it is different for every organization, since every organization is different.

Nobody likes pain and difficulty; many of the impediments are so inherent to an organization's way of thinking and operating that they are very difficult to remove. No amount of planning up front will mitigate this difficulty; it will only help alert everyone to the hard work that must be done to become a world-class competitor. Scrum requires that senior management be vitally involved in impediment triage and removal, and therefore requires that the CIO adopting Scrum become the *leading agent for change*.

In this way, the CIO engages in a process of continued organizational improvement, all aimed at increasing the productivity and quality of the software teams. It's not easy, and the leadership the CIO provides will be a critical factor in success, as the following note from Ken Schwaber to a CEO illustrates:

From: Ken Schwaber
To: XXX XXXXX, CEO for XXXXXXX Corporation

"On one hand, Scrum offers some very attractive possibilities – increased productivity, a better working environment, increased competitiveness, and a higher quality product. On the other hand, it is hard to implement. The amount of change engendered by a Scrum implementation is significant and difficult.

Even though the change is difficult for the developers and customers (product owners), they have immediate payback through increased job satisfaction. This helps them through times of stress and anxiety. Middle management, however, is stressed without immediate reward. They are asked to help transition an organization from traditional approaches to leaner approaches without a clear vision of a personal end point … what will I do and where will I fit into the new organization. This question is particularly difficult and fraught with danger since middle management will be fashioning the new organization. The potential for conflict and politics is daunting.

My experience with top-down, enterprise implementations of Scrum has led me to believe that the differentiator between success and failure is you. Your ability to vision the future and help communicate it to your management, your ability to patiently guide them through the change, and your ability to assure your middle management of their value and form them into a team will differentiate your ability to absorb the change and realize the benefits, or not."

A Playbook for Adopting Scrum

Once you decide to implement Scrum within your organization, a journey begins with a belief that the effort will be rewarded with a more effective software process and a more responsive and competitive company. It also recognizes that a significant amount of organizational change is now in the forecast.

As the CIO contemplates this undertaking, an understanding of organizational behavior leads to a rational set of steps for achieving substantive change. These include:

- Finding an evangelist and local sponsor;
- Taking small initial steps that test the waters;
- Reflecting on successes and failures, then moving forward, step by step.

This next section describes some typical examples of how you might implement Scrum throughout your organization; a "playbook" that gives sample techniques you can apply to accomplish the requisite change.

Play 0 - Overview, Assessment and Pilot Preparation

The objective of the first play is to prepare the playing field for the activities ahead by a) assessing the organizations readiness for agility, b) providing initial training for the early participants and c) building the Product Backlog for the initial projects. The details of this play are as follows:

Overview and Assessment

Description: Two day working session consisting of

- Scrum Aptitude Test – exposes management to the types of change that happen with Scrum, and helps them determine if they want to proceed.
- Scrum presentations – raise general awareness and present concepts to entire organization.
- Assess organizational readiness and define next steps.
- Define plans; identify potential pilots, schedule training, and resource the pilot project.
- Dinner with senior management to review next steps.

Duration: 2 days

Support: External

Pilot Preparation

The organization is ready to proceed with the training and structure needed to support the first pilot project. Activities in this phase include:

ScrumMaster training

Description: Train ScrumMasters to run the pilots

Duration: 2 days

Support: External

Product Owner training

Description: Train Product Owners to maximize ROI using Scrum.

Duration: 1 day

Support: External

Establish Metrics

Description: Review and modify metrics that monitor the use of Scrum within the organization and define the value derived from the pilots. Establish core Scrum process and project metrics.

Duration: 1 week

Support: External

Establish Change Product Backlog

Description: Establish product backlog for tracking and evaluating impediments that arise during the pilot projects. This will be the basis for change action within the organization.

Duration: 1 day

Support: External

Play 1 - Pilot Project(s)

The objective of this play is to experience Scrum on one or more real projects in order to demonstrate the positive benefits of improved software agility within the organization. One or more pilot projects are now executed. ScrumMasters and management closely watch the pilots to identify organizational obstacles and impediments to Scrum. When these impediments are identified, they are fixed on the spot where possible, or are simply recorded in the Organizational Change Backlog and categorized for later attention.

Pilot projects

Duration: 3-6 months

Support: External / Internal ScrumMaster

Description: Run 3 to 6 iterations of the pilot projects. Pilot projects deliver increments of functionality and identify impediments to optimized software development. Assess and adjust plan, evaluate and prioritize impediments.

Retrospective

Duration: 2 day

Support: External / Internal ScrumMaster

Description: Review pilot projects, metrics, and impediments. Assess what went right, what could be improved. Identify the ROI. Assess impact on business operations, including relationships within organizational departments and with customers.

Re-planning

Duration: 1 day

Support: External / Internal ScrumMaster

Description: Modify master plan for Scrum implementation; keep it high level and let project plans and the organizational change plan be driven by their own specific product backlogs.

Play 2 - Organizational Expansion

Based on successful pilots, the objective of this play is to expand the usage of Scrum and its benefits to a significant subset of the development organization. By now, there is an understanding of what beneficial practices are embedded, what impediments stand in the way of broader adoption and where further training is required. For example, the following broader training programs may now be effective:

- ScrumMaster training: Before scaling the implementation to additional and larger projects you must be increase the number of Scrum Masters. Candidates with appropriate skills should now be recognized in the organization. ScrumMasters who will be leading Scrum of Scrums (see below) can now be trained in advanced skills like Team Facilitation and Metrics Collection.
- Product Development Training: We optimize the hand-off between analysts and development teams when these two roles use a common approach. An often seen approach is to adopt Lean or Toyota style product development methods. The references list a number of books that provide information on these topics.
- Engineering Training: The engineering teams involved in agile projects will have learned where they need skills to operate in a more agile manner. Training in Extreme Programming (XP) skills like Test Drive Development etc. may now be warranted. (Beck 2004)
- Scrum/Agility training: a successful implementation of Scrum will largely depend on a common vocabulary of all people involved. This can be achieved through 2 – 4 hour introduction courses for 30-50% of the organization.

In addition, you may apply other activities to increase the visibility and level of acceptance of Scrum in the organization:

- Information radiators: Communicate the state of Scrum projects through simple and powerful information radiators, like whiteboards showing the tasks (Task Board), Product and Release Backlogs and the project and program BurnDown charts.
- Reading: A suggestion of articles and books can be provided to all people in the organization to encourage further knowledge expansion.
- CIO led seminars/brownbags: The change leader(s) should communicate often and openly about what is happening in the organization. Informal meetings, like brownbags and pizza hours tend to have a positive impact on change.
- Chats/war stories/feedback from the pilot(s): The results from the pilot projects should be available to everybody. This will increase discussion and involvement through all levels of the organization.

Play 3 – Achieving Impact

As the pilot projects have proven that real value will be delivered through an agile approach to software project management, the objective of this play is to achieve a more significant impact on the bottom line which can only be demonstrated through more and larger projects. Through the previous plays the organization has collected sufficient explicit and tacit knowledge to be able to tackle these with a high probability for success. At this point, as much as 25% of the organization should be involved in the implementation of Scrum.

Effective change should now be occurring inside and outside the development organization. Inside development, the work is best done by the development team. Outside the development teams, the work of eliminating impediments is directed by the Organizational ScrumMaster and is implemented by the affected departments.

Development Projects

Duration: Forever

Support: Internal

Description: Development projects monitored by ROI.

Change Projects

Duration: Most work in first 1 to 2 years; then, as needed

Support: Internal

Description: Organizational change projects within various departments drive out emerging and changing impediments.

Assess and Adapt

Duration: Every Sprint

Support: External / Internal ScrumMaster

Description: Review qualitative and quantitative metrics. Add additional metrics and review capture processes whenever a surprise has occurred.

Play 4 - Measure, Assess and Adjust

The objective of this play is to assess the organization's progress and to establish a broader set of metrics to serve as a basis for further expansion. The CIO should be aware that the upcoming discussion of metrics may be both controversial and entertaining as many of the traditional metrics that might be in place prior to Scrum adoption (example: measures of "document completeness") are no longer relevant. Fortunately, Scrum and agile practices are indeed accountable and measurable and practitioners are converging on a set of metrics that provide qualitative and quantitative feedback at both the process and project level.

But before entering this discussion, a *key distinction* needs to be made between many traditional software development processes and Scrum and agile:

> *The primary metric for agile software development is whether or not working software actually exists, and is demonstrably suitable for use in its intended purpose. In Scrum, that key indicator is determined empirically, by demonstration,* ***at the end of every single Sprint****.*

This primary measure of software quality and productivity is the essence of agile development. So with Scrum, you cannot be very far off your objective without knowing that you are. All other metrics are subordinate to that objective and its constant mantra of *"delivering working software more frequently"*.

At this point in the Scrum adoption game, a significant part of the organization is now operating in an agile manner. Sprint results of the initial projects are the primary measure of the effectiveness of the new team behaviors and their new processes. This data should be published and analyzed.

Moreover, now is the appropriate time to define a set of secondary metrics used to guide your organization on how it implements Scrum. In so doing, there are two types of metrics that may be applied:

Process Metrics – primarily qualitative indicators on the effectiveness of the teams and organization in adopting Scrum. These include items such as effectiveness of the teams in managing the Product Backlog, effectiveness of Scrum processes such as the Scrum daily meeting, Sprint Planning meeting, etc. In cooperation with the ScrumAlliance, Hartman and Stallings have developed a template of process metrics that can be used by any organization that is implementing Scrum. These metrics appear as Table 1 in Appendix 2.

Project Metrics – At the project level, an additional set of metrics may be applied to measure the results for a particular Scrum team and the service, component or system that they are accountable for. These may include some traditional metrics such as defect count, percentage of code with unit test coverage, percentage of code covered by automated regression tests, etc., as well as Scrum specific metrics such as number of user stories finished and demonstrable at the end of each Sprint. An example set of these metrics appears as Table 2 in Appendix 2.

A Note on Quality and Scrum

Customers often pressure development organizations to deliver features faster than is feasible. Some organizations accommodate this by reducing the quality of the product, dropping re-factoring, cutting test efforts and other solid engineering practices. This is not supportable within Scrum practices since the system or product is a corporate asset, refined continuously and objectively measured, not a one-time project asset. Engineering organizations that succumb to this pressure eventually build "design dead" systems that can not be effectively maintained or enhanced. The organization suffers the huge cost of a substantial rewrite and re-

release of the code base. To avoid this, only the senior levels of an organization can make an asset decision for reducing quality.

Play 5 – Expand and Win

With these activities behind the organization, and with a defined set of metrics to guide and evaluate future progress on an organization-wide basis, it is now time to expand the use of Scrum across the entire organization. The activities in this phase of the implementation are focused on the further scaling of Scrum within the organization.

In steps of perhaps 25–30% of head count, the remaining teams in the organization are introduced to Scrum. Existing practices are further refined and shared between the teams in order to reach an organizational inculcation of the agile practices. Only now can the strict rules with which Scrum operates be adjusted to better match the need of the organization. Customers can be invited to participate in the implementation through training as product owners or ScrumMasters. This phase will continue until all teams are involved in Scrum, and Scrum's inspect-and-adapt mechanisms will address further enhancement of your processes and practices.

At this point, the organization will be receiving the substantial productivity and business and cultural benefits of Scrum.

Before we proceed to Scaling Scrum to the largest project environments, however, we need to look at the types of organizational impediments that can prevent effective Scrum practices.

Organizational Impediments to Adopting Scrum

Applications developed in any organization are intended to optimize the ability of the company to meet its business mission. However, over the course of time, organizations evolve in ways which are not always conducive to the productivity of the software team that develops and maintains those applications. Indeed, some organizations have evolved to the point where the software practices are largely dysfunctional and – despite repeated efforts to improve them – the organizational structure, policies and strictures prevent effective change. This section describes the source and nature of these impediments to better arm the CIO for the work ahead.

Exposing the Impediments with Scrum

The very nature of Scrum; its incessant demands for quality software to be delivered more quickly; its continuous demand for working with end users to assure effective implementation, and its continuous inspection and adaptation mechanisms expose dysfunctional practices and "blocking issues" very quickly. This effect becomes all the more pronounced when Scrum is also used as a process to implement and scale Scrum in the organization.

It's impossible to identify all the organizational efforts up front

You cannot identify all impediments up front as they are embedded in the organization and therefore too familiar to be identified easily. Only when you start using Scrum do they become obvious. The plan for implementation emerges as the evidence of what needs to be changed and the organization's *willingness to make the change* emerges.

Characterizing Impediments

Impediments will generally be encountered in four areas:

Scrum Process Itself – what impediments are occurring that get in the way of the Scrum process?

People Practices – what people practices are getting in the way of developing, distributing, supporting and using products to maximize the fulfillment of everyone involved?

Product Engineering Practices – what practices are impeding the optimization of return on investment, or maximizing the mission of the organization from a product perspective, and what impediments are there to optimized product development and delivery?

Organizational Issues – what systemic organizational issues - that lie clearly outside the team's control - are preventing the teams from delivering software to its users more quickly?

We want separate categories in the Organizational Impediment Product Backlog because these require unique skills to resolve. In addition, they should be prioritized as to impact, and some thought should be given as to who best in the organization can best resolve the impediment.

Table 1 below shows examples of impediments found in organizations adopting Scrum. This list might provide a starting point and an early warning system for some of the impediments your organization is likely to encounter. But again, every company is different and every implementation is different, so fortunately or unfortunately, the Scrum process will assuredly discover some new impediments for the CIO to deal with!

	Impact	Cost	Owner
Scrum Process			
People arrive late to daily Scrum and do not support basic discipline			
Scrum meetings take too long – team is bored and considers the time unproductive			
ScrumMaster dictates design decisions or micromanages			
Teams are too large for effective daily Scrum and Sprint planning			
Teams do not report task remaining time for BurnDown analysis			
People Practices			
Individuals interrupted and tasked to work outside the Sprint			
Teams isolated in cubicle and not in open Scrum area			
Team members not accountable for personal Sprint commitments			
Individuals are multiplexed across too many projects and teams			
Product Engineering Practices			
Cross-functional resources for definition, design, implementation and test are not present on the team			
Sprints do not fully implement and test potentially deployable increments of customer valued features			
Product owner not easily available/not integral to team			
System integration is not forced at each Sprint			
Product owner won't split up massive product backlog items to fit within a Sprint.			
Teams have ineffective resources for automating builds and integrations			
Features loaded into Sprint after Sprint begins			
Organizational Issues			
Software process police regulate to ineffective processes			
Management assumes fixed price, fixed time, fixed functionality delivery posture			
Software Test/and or System QA is separate organization and is not integrated with team			
Organization rewards individual, rather than team behavior			
Existing rules or software capitalization demand adherence to document-driven, waterfall approaches			
Teams not co-located to maximum extent feasible			
Teams cannot make small organizational, space and expense decisions needed to do their job			

Legend: **Impact** of this impediment to the project (0-9, 0 low, 9 high),
Cost to resolve this impediment (0-9, 0 low, 9 high)
Owner: Point in an organization for resolution: C – CEO/CTO/COO/CFO, V – VP, D – Director., P – Product Management, E – Engineering Management, T – Team

Scaling Scrum

The business benefits of Scrum and agility are most readily achieved with small, co-located and integrated teams, ideally consisting of eight people or less (including Product Owner, ScrumMaster, Developers and Testers) and where each Scrum team owns a specific product or application that they can define, develop, test and deliver without much outside help.

Inevitably, however, the success of Scrum will lead to its application to larger programs, systems of systems, and applications that take many, likely distributed, teams to develop and deliver. Fortunately, Scrum has been proven in projects consisting of many hundreds of developers so Scrum does scale to the challenge of the larger software enterprise. Doing so, however, brings about a unique set of challenges that must be addressed, specifically:

1. Scaling the organization: Scrum teams of teams
2. Tooling infrastructure for enterprise agility
3. Coordinating teams of teams

Each of these challenges is addressed in the sections below.

Scaling the Organization: Scrum Teams of Teams

Consistent with its less is more philosophy, Scrum has a very small number of rules. However, most of the rules that do exist are fixed and relatively inviolate. One basic rule is the team consists of eight or fewer members that are co-located in a common seating area. This is the most effective and productive model as it a) supports the requirement for constant informal communication amongst the team members, b) fosters a high degree of esprit de corps and c) allows for a mutual commitment to the goals of the Sprint amongst team members who actually know each other and have to work together every day. In addition, certain Scrum mechanisms, such as Sprint planning and the Daily Scrum meeting can breakdown very quickly as the team size gets beyond 8-10 individuals.

Scaling Scrum to larger applications leaves this key principle in place. So, scaling to an application involving 300 people involves organizing around 30 Scrum teams. As previously discussed, the team's complement must be fully rounded and capable of developing potentially shippable pieces of functionality at every Sprint. For most organizations, this requires reorganizing teams around product features, services, components or subsystems, rather than by individual role (e.g. developer pool, test resources, etc.). While we discussed this organizational impediment earlier, we see it gets compounded as our project's size increases.

Organization Follows Architecture

Moreover, we cannot readily form Scrum teams without understanding how each individual team can relatively holistically deliver end user functionality. In turn, this mandates that we decompose the application architecture into components or subsystems that have conceptual integrity and can deliver business value on their own[1]. Scrum provides for this architectural factoring activity in the Sprint staging phase, and in early Sprints, by the front-running Scrum teams. This method works particularly well in a period of Scrum expansion and rollout for a large project. Here, the front-running teams build proof points of customer value while they simultaneously factor

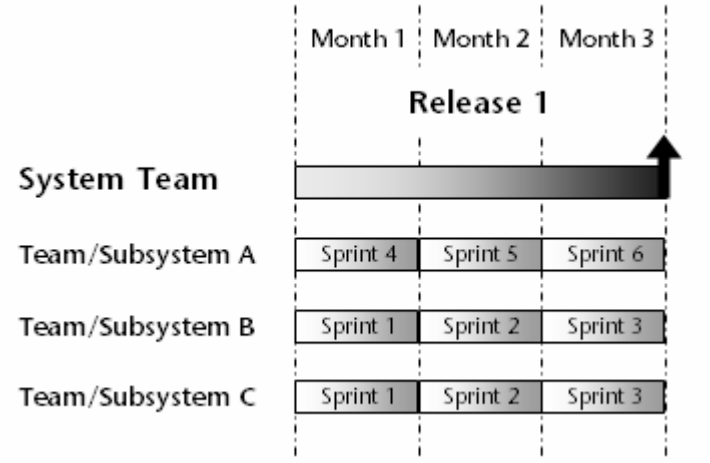

Figure 2: A System Being Built by Three Scrum Teams Over Three Sprints

[1] This level of sharing and communication can indeed be a challenge when implementing service oriented architectures, as the existing organization likely mirrors the prior architecture of independent silos of applications whose department owners were not overly-required to cooperate to deliver more flexible services to users, as now becomes the case.

the application architecture to accept additional teams, whose Scrum training will likely be occurring at about the same time. As each new team is formed, its role in the larger system becomes clear and a picture like Figure 2 emerges.

Coordinating Teams of Teams

Of course, the presence of a large number of teams brings significant challenges in coordinating and communicating across teams, and also implies that there will likely be a number of issues at the system level which require the same daily and monthly inspection practices applied at the local team level. Experiences with scaling Scrum to larger teams have evolved a small set of useful practices for coordinating disparate teams and addressing the larger challenges of Sprint planning, release planning and tracking system level integration and test activities.

Daily Communication: Scrum of Scrums

In the same fashion that Scrum mandates daily communication in the daily Scrum, larger and distributed teams typically coordinate their activities in a daily Scrum of Scrums. In this meeting, team leaders from each component team use the same format as the single team daily meeting:

1. What did my team do yesterday to advance the objectives of the Sprint?
2. What will my team do today?
3. What impediments are present that could keep my team from meeting its commitment to the Sprint?

Ideally, this meeting occurs immediately after the individual team's daily Scrum. When teams are dispersed, it often occurs by telephone with the time of day selected to maximize participation amongst the scrum of scrum team members.

System Level Release Planning and Tracking

Figure 2 might imply that it is a fairly straightforward matter to divide the organization into feature, service or subsystem teams, empower these teams to do their jobs, and that a wonderfully integrated system will naturally occur. Experience has shown that this is unlikely. For even when the individual teams are empowered to meet both the needs of the Sprint and coordinate integration between the teams/subsystems, a larger set of challenges is present. That is the challenge of building a system holistically, where we implement ant test our integrations across all subsystems, where subsystems work together to meet broader customer requirements and that the overall system meets its quality, performance and reliability requirements.

To address these challenges, many teams have added a technical lead role played at the system level. Architects, team leads, product managers and quality assurance personnel will often grow into an additional Scrum team to think and act at the system level. Moreover, they can also apply the Scrum process at the system level to set Sprint objectives and create backlog items of forced system integrations, system level demonstrations, quality checkpoints, trial distributions and other milestones to assure that the system stays on track. In so doing, the picture in Figure 3 starts to emerge.

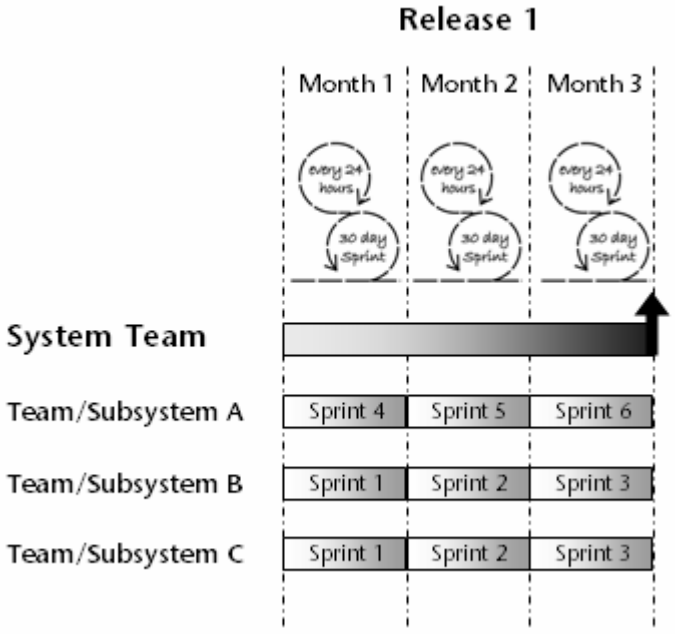

Figure 3: System of Three Subsystems with System Level Sprints

Tooling Infrastructure for Enterprise Agility

Even with this level of structure and coordination, larger projects and distributed teams may still find themselves lacking the internal and cross-team coordination and project visibility required to reliably deliver software in rapid, fully-tested iterations. While Scrum provides a proven framework for the project management aspects of software development, it does not prescribe specific software engineering practices nor recommend specific tooling to support the Scrum process. Scrum's philosophy in this regard is "keep it simple and let the teams decide."

Indeed, for the ideal team of less than ten co-located persons, the prime project management artifacts used to plan the Sprint and communicate status of individual features, tasks and team progress can often be managed using a spreadsheet developed and maintained by the ScrumMaster. The engineering artifacts for requirements, test cases and defects may be equally lightweight and written on index cards, whiteboards or maintained on a team wiki.

People and Communication

Scaling Scrum presents special tooling challenges

However, scaling Scrum practices to distributed teams, and teams of teams, presents special communication challenges. Cross-team coordination of how to implement shared requirements, track feature status and identify blocking issues becomes a primary concern. In these cases *"a mechanism for frequently synchronizing their work must be devised and implemented. Also, a more detailed product and technical architecture must be constructed so the work can be cleanly divided across teams."* (Schwaber 2004)

While traditional project management tools may have worked for showing idealized task start/stop dates and performing - perhaps fruitless - critical path analysis on long waterfall projects, these plan-driven activities lose their relevance when working in short iterations where the entire team focuses on driving the few highest priority features to acceptance. Instead of one person maintaining a separate task database that is decoupled from the day-to-day artifacts the team is actually planning and implementing (e.g. user stories and tests), larger programs need a real-time collaboration environment that supports the natural signaling occurring among team members as they advance a feature from the Product Backlog into development, testing and integration. To emulate the co-located team, this agile project management environment must let everyone quickly see and update where a feature is in its lifecycle, how much effort remains before its completion and what specific issues are blocking its progress.

Besides needing new ways to plan and track our iterations, the capabilities of tools applied to defining, organizing and sharing our system artifacts have new demands as well. Managing requirements, their acceptance tests and defects calls for support that is horizontal across the lifecycle activities inside a Sprint, not vertical with deep silos of artifact information that are poorly related to the commitments the teams have made. In fact, with rapid iterations, it is really the relationships between these artifacts that are the primary concern to the teams. After all, each Sprint is producing many pieces of working, tested code, so the teams must understand exactly how these engineering artifacts relate to each other and be able to see their status at every point in time.

Tooling Infrastructure Opportunities

Being software developers after all, the teams will naturally want to better organize their artifacts and automate those aspects of the Scrum process that lend themselves to software support. Specifically, the teams will likely want to add infrastructure support for the following activities and artifact types in the software lifecycle:

- **Backlog Management** – As system complexity grows, the team will want better support for capture and maintenance of the feature lists, functional and nonfunctional requirements, use cases, and user stories as well as the priorities, estimates, status and owners of these items. As Scrum is applied to larger projects, these artifacts may grow to many thousands in count and a means to organize, support and view them by system or subsystem becomes critical.
- **Project Reporting** – Scrum eschews traditional, waterfall-like project *plans*, but the tactical day to day *project management* nature of Scrum is intense and unremitting. The team will need a simple way for each member to enter their task estimates, status, and effort remaining so that the BurnDown Charts are automated and continuously available. In addition, the infrastructure should support the natural signaling teams use as backlog items move through their lifecycle. Senior personnel will need to look across teams and understand their individual iterations and release plans in order to asses the status of their program as a whole.
- **Just-in-Time Requirements Elaboration** – Many smaller Scrum projects succeed with informal requirements mechanisms such as direct discussion between the Product Owner and Team, but as project complexity and criticality grows, more depth and richness of requirements expression and requirements versioning will likely be required. For example, documentation of interfaces that affect multiple teams becomes critical. Changes to interfaces or new features that cross team boundaries may have a significant impact on the project. These requirements should be elaborated on a just-in-time basis, meaning at, or just prior to the Sprint that implements the new functionality. To address this problem, teams may want centralized support for richer forms of requirements expression, their compilation for review and automated change notification.
- **Early Testing:** Since every Sprint delivers potentially shippable code into the product baseline, early test case development and test automation enables teams to support the rapid iteration requirements of Scrum. Tooling that generates test cases directly from requirements or story cards will accelerate the development process and provide the inherent traceability needed to prove acceptance of the feature. Know that the ongoing management of the hundreds and thousands of regression tests that accrue will likely become the critical factor in determining the speed and success of your Sprints.
- **Release Planning** – The philosophy of Scrum focuses on the “art of the possible in the nearer term”, as opposed to the black art of supposedly predicting exactly what will be delivered 6-12 Sprints down the road. This philosophy is a breakthrough in thinking at the team level because it allows Scrum teams to focus in a “heads down” fashion for 30 days at a time, and thus produce working software more reliably. But as the teams grow and scale, applying additional analysis and rigor to Sprints beyond the immediate horizon helps avoid architectures that require substantial re-factoring down the road. While re-factoring is highly encouraged in agile, it becomes less practical as the scope of the application and the number of existing deployments increases. Additional release planning that provides us with architectural runway is often appropriate. Therefore, the art of Sprint planning can include “a few Sprints out” and “what-if” planning functions that help the teams make backlog tradeoffs and communicate a reasonable vision and product roadmap to the sponsors.

In addition, these teams will typically want to organize all these assets in a central repository where every team member can access them, *24x7, worldwide*, and one which provides instantaneous views of project and program status, with automated change notification for critical project changes.

Evolving the Infrastructure in Sprints

In Scrum, deploying this level of infrastructure is not a one time event, done "up front" by a tooling team. Instead, the Scrum teams themselves take on the task of identifying what they will buy and build to address their problems based on the lessons learned in prior Sprints. Moreover, these investments are made in the context of ongoing Sprints. Therefore the team addresses the build out of infrastructure by adding items to the Product Backlog to address the infrastructure items as per Figure 4 below. Of course, customer facing functionality still takes priority, but the experienced team recognizes they must be able to continuously schedule infrastructure work as well in order to maintain their velocity and productivity as the application scope and number of teams grows.

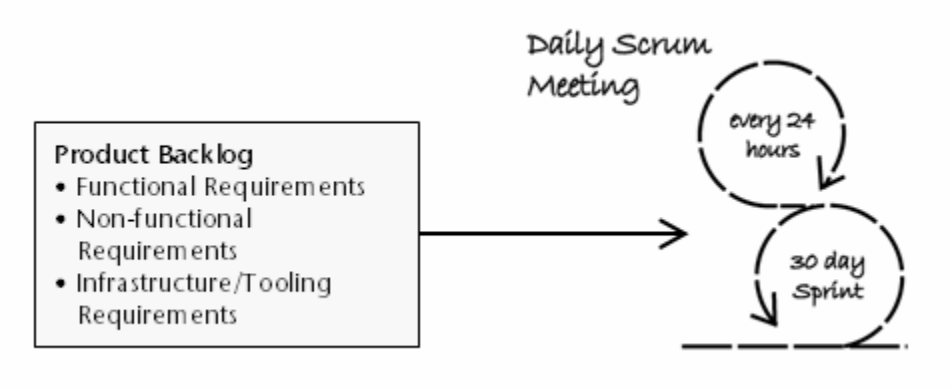

Figure 4: Parsing Scalability Infrastructure into a Sprint

Summary

Scrum is a proven and effective software development practice that can rapidly increase the productivity, delivery speed and quality of software teams.

What IT organization would not benefit from these attributes commonly experienced by successful Scrum implementations?

- Decreased development cycle times
- Higher value throughput to end users
- Higher quality
- Lower development risk
- Greater user satisfaction
- Improved company morale

While appearing simple on the surface, implementing Scrum often requires substantial organizational change to eliminate the impediments to effective development and delivery. As lead change agent, the CIO or other executive sponsor has the primary responsibility for eliminating these impediments. It is the enduring commitment by the CIO that may well be the difference between success and failure of the implementation. While none of this is easy, the CIO who commits to improving software outcomes with Scrum will take the first step in ensuring that the enterprise is well on its way to achieving the business benefits of faster and better quality software delivery.

In addition, Scrum is highly effective in large-scale enterprise application development and can support the needs of many hundreds of developers working on shared applications. Scaling Scrum presents an additional set of challenges to infrastructure and tooling that the teams themselves will address – but overcoming these challenges will likely deliver a substantial advantage to these larger organizations over their marketplace rivals.

Bibliography

Agile Manifesto: www.agilealliance.org

Beck, Kent. Extreme Programming Explained: Embrace Change (2nd Edition). Boston: Addison-Wesley, 2004.

Schwaber. Ken. Agile Project Management with Scrum. Redmond, WA: Microsoft Press, 2004.

Schwaber, Ken, and Mike Beedle. Agile Software Development with Scrum. Upper Saddle River, N.J.: Prentice Hall, 2002.

Appendix A - Resources

Selected Reading

Beck, Kent. Extreme Programming Explained: Embrace Change 2nd Edition. Boston: Addison-Wesley, 2004.

Cockburn, Alistair. Agile Software Development. Boston: Addison-Wesley, 2002.

Cohn, Mike. User Stories Applied. Boston, MA: Pearson Education, 2004.

Highsmith, Jim. Agile Project Management: Creating Innovative Products. Boston: Pearson Education, 2004.

Nonaka, Ikujiro and Hirotaka Takeuchi. The Knowledge-Creating Company. Oxford University Press, 1995.

Poppendieck, Mary and Tom Poppendieck. Lean Software Development. Addison-Wesley, 2003.

Schwaber. Ken. Agile Project Management with Scrum. Redmond, WA: Microsoft Press, 2004.

Schwaber, Ken, and Mike Beedle. Agile Software Development with Scrum. Upper Saddle River, N.J.: Prentice Hall, 2002.

Larman, Craig. Agile and Iterative Development: A Manager's Guide, Boston: Addison Wesley 2004

Agile Training

SCRUM Master Class - www.controlchaos.com/certifiedscrum/

EXtreme Programming - www.xprogramming.com/xpmag/services.htm

Agile Tooling and Training

Rally Software Development: www.rallydev.com

Other

Agile Alliance: www.agilealliance.org

ScrumAlliance: http://www.scrumalliance.org/

Possible newsgroups for Scrum / Agile interested people are:

Scrum Development: http://groups.yahoo.com/group/scrumdevelopment/

Agile Management: http://groups.yahoo.com/group/agilemanagement/

Agile Testing: http://groups.yahoo.com/group/agile-testing

Lean development: http://groups.yahoo.com/group/leandevelopment/

Appendix B - The Agile Manifesto

In 2001, a workshop was held in Snowbird, Utah, USA, where various originators and practitioners of agile methodologies met to understand what they had in common. They picked the word "agile" for an umbrella term and crafted the Manifesto for Agile Software Development, whose underpinning are a statement of shared values:

> *"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:*
>
> - Individuals and interactions over processes and tools
> - Working software over comprehensive documentation
> - Customer collaboration over contract negotiation
> - Responding to change over following a plan
>
> *That is, while there is value in the items on the right, we value the items on the left more."*

The Manifesto struck a chord, and it led to many new agile projects being started. As with any human endeavor, some succeeded and some failed. But what was striking about the successes was how much both the business people and the technical people loved their project. This was the way they wanted software development done. Successful projects spawned enthusiasts.

Principles behind the Agile Manifesto

The Agile Manifesto signatories agreed on twelve principles that underwrite and re-enforce the manifesto. These principles focus on deliverables and change, people & communication and feedback:

Deliverables & Change

Instead of prescribing or advising on large number of artifacts that should be delivered as the result of a successful project completion, agile methods go back to the principle deliverable: working software.

- Working software is the primary measure of progress.
- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- Simplicity -the art of maximizing the amount of work not done- is essential.
- Continuous attention to technical excellence and good design enhances agility.

People & Communication

Building working software systems is first and foremost about people.

- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- Business people and developers must work together daily throughout the project.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Feedback

No process or method can be static, hence feedback and self-adjustment needs to be build-in.

- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Credit to agile leaders

A number of thought leaders have participated in the development of various agile methods, while impossible to recognize them all, these include: Kent Beck (XP), Mike Beedle, Ken Schwaber & Jeff Sutherland (Scrum), Alistair Cockburn (Crystal), Ward Cunningham (FIT), Martin Fowler (XP), Jim Highsmith (Agile Project Management), Marie & Tom Poppendieck (Lean Software Development) and Ron Jeffries (XP).

Appendix C - Scrum Metrics

Table 1 – Scrum Process Metrics (Hartmann, Stallings)

Project:		
Scoring (0-5)	**Sprint 1**	**Sprint 2**
1. Product Owner		
Product Backlog developed owned and managed by Product Owner		
Product Owner process is flexible; collaboration with team is ongoing		
Product Owner and stake-holder participation at Sprint Review		
Product Owner manages project by value		
Total "Product Owner" Score		
2. Planning		
Product Backlog is descriptive, prioritized and has effective estimates		
Team develops and manages Sprint Backlog		
Team involves stakeholders and dependencies in effective manner		
Project progress can be tracked by backlog BurnDown and value burnup		
Sustainable pace		
Total "Planning" Score		
3. Schedule		
Sprint Planning regular, on time, fully attended		
Sprint Review regular, on time, fully attended		
Daily Scrum occurs on time, is fully attended		
Team meets its commitments to Sprint		
Total "Schedule" Score		
4. Process		
Team self-polices and reinforces use of process and rules		
Organization is able to comply with Scrum rules		
ScrumMaster is effective in getting process followed		
Team is self-managing		
Surprises don't occur		
Team is cross-functional		
Team and Product Owner collaborate and work closely together		
Team works to improve itself and its processes		
Team adequately manages dependencies		
Total "Process" Score		

Project:		
Scoring (0-5)	**Sprint 1**	**Sprint 2**
5. Team		
Team members are dedicated and honor commitments		
Team effectively acts upon indicators in Sprint Burndown		
Communications between team members is effective		
Team effectively manages conflict within team		
Team improves internal development processes		
Total "Team" Score		
6. Reporting		
Product Backlog is effectively maintained and communicated		
Sprint Backlog is effectively maintained and communicated		
Project and Sprint reporting are effective and understood		
Value is used to manage Product Backlog		
Total "Reporting" Score		
7. Engineering Practices/Infrastructure		
At least daily build		
Common source code library		
Metric for quality of increment		
Increment metric met		
Test driven development practices		
Refactoring practices		
Design review practices		
Code review practices		
Coding standards		
Automated unit test harness		
Automated acceptance test harness		
Total "Engineering" Score		

Table 2 – Scrum Project Metrics (Rally Software Development Corp.)

Project:		
	Sprint 1	**Sprint 2**
Functionality		
New Features Planned		
New Features Completed		
Story cards		
In iteration		
accepted		
% Accepted		
# not accepted		
# pushed to next		
# pushed later		
# added/deleted		
Quality & Test Automation		
% SC with test available / tests automated		
Open Defect Count (P1 + P2)		
Total Open Defect Count		
# test cases		
# manual test cases required to regress		
Code coverage %		
Architecture		
Refactors completed		
Refactors in progress		
Refactor backlog		
Customer debt features		
Customer feature debts accepted		
Release Plan		
Confidence that release plan is understood		
Confidence in achieving plan		
Planned Release Date		
Actual Release Date		

