Scrum Methodology

Incremental, Iterative
Software Development
from Agile Processes

Revision 0.9

Table of Contents

Subject: ***License and Restrictions***

Notice

This copy of Scrum is governed by the terms and conditions of an ***Individual Scrum License***. Such license must be signed by a representative of Advanced Development Methods, Inc., be within the license term, and be in the possession of the Licensee for the use of this copy of Scrum to be legal. This license has been granted to a specifically named individual who is a Certified ScrumMaster, as listed at http://www.controlchaos.com/certifiedscrum. Users, viewers, readers, and others seeing this copy of the methodology must be that specifically named individual. All other use by any other party is not covered by this license.

The License has the following restrictions:

1. The Licensee cannot share the Scrum methodology with any other parties or make copies other than for backup and never for the purpose of sharing or distributing.. The Scrum methodology is for their individual use as a reference source only;
2. The Licensee cannot resell, relicense, or transfer ownership of this license in any manner to others; and,
3. The Licensee cannot let anyone else use the materials that are part of the License.

Subject: License and Restrictions

Scrum License (Licensee Copy)

This Individual License for Scrum (the "License") is granted to

_____ ("Licensee") in consideration for the Licensee's successful completion of the ScrumMaster Certification class held _____,2003.

In fulfillment of this License, **Licensee is provided with**:

1. Scrum methodology, as a PDF;
2. Listing as a Certified ScrumMaster on the Certified ScrumMaster website;
3. Scrum software, as a Microsoft Excel 2000 add-in;, and,
4. Scrum training materials as used in the class

The licensee is entitled to use all of the materials in the License for their own individual use. For example:

1. The Scrum methodology can be used as reference material;
2. The Scum training materials can be used to train others in the Scrum methodology; and,
3. The Scrum software can be used by the licensee to develop backlog and track project progress.

The License has the following restrictions:

1. The Licensee cannot share the Scrum methodology with any other parties or make copies other than for backup and never for the purpose of sharing or distributing.. The Scrum methodology is for their individual use as a reference source only;
2. The Licensee cannot resell, relicense, or transfer ownership of this license in any manner to others; and,
3. The Licensee cannot let anyone else use the materials that are part of the License.

This License is valid for two years. It will expire two years from the date of issuance. If Licensee wishes to renew the License at that time, facilities for relicensing will be posted at the Certified ScrumMaster website.

For Advanced Development Methods

Name: Date:

For Licensee

Name: Date:

Email Address:

Subject: License and Restrictions

Scrum License (ADM Copy)

This Individual License for Scrum (the "License") is granted to

_____ ("Licensee") in consideration for the Licensee's successful completion of the ScrumMaster Certification class held _____,2003.

In fulfillment of this License, **Licensee is provided with**:

1. Scrum methodology, as a PDF;
2. Listing as a Certified ScrumMaster on the Certified ScrumMaster website;
3. Scrum software, as a Microsoft Excel 2000 add-in;, and,
4. Scrum training materials as used in the class

The licensee is entitled to use all of the materials in the License for their own individual use. For example:

1. The Scrum methodology can be used as reference material;
2. The Scum training materials can be used to train others in the Scrum methodology; and,
3. The Scrum software can be used by the licensee to develop backlog and track project progress.

The License has the following restrictions:

1. The Licensee cannot share the Scrum methodology with any other parties or make copies other than for backup and never for the purpose of sharing or distributing.. The Scrum methodology is for their individual use as a reference source only;
2. The Licensee cannot resell, relicense, or transfer ownership of this license in any manner to others; and,
3. The Licensee cannot let anyone else use the materials that are part of the License.

This License is valid for two years. It will expire two years from the date of issuance. If Licensee wishes to renew the License at that time, facilities for relicensing will be posted at the Certified ScrumMaster website.

For Advanced Development Methods

Name: Date:

For Licensee

Name: Date:

Email Address:

Subject: Introduction

Scrum is simple. The practices are few and straightforward. But Scrum is hard. Management must care and constantly be tending to the project realities rather than plans.

The Scrum methodology consists of this process description and the Scrum Project Management software, which provides automated support for some of these activities.

The Scrum methodology is a complete methodology for managing the development of products. Scrum is completely scalable, from small to large projects, from simple to complex projects. Scrum is an agile process, so the probability of success with Scrum stays high throughout levels of complexity until chaos is reached, at which point no methodology or process is adequate.

The Scrum methodology consists of phases and activities. The phases are Planning, Staging, Development, and Release. Planning and Staging prepare the workload for the Development phase, where all functional development is done iteratively, each Sprint creating a complete increment of potentially releasable product functionality. The methodology spells out the few artifacts and roles that are required to manage a project. These artifacts and roles are grouped within the phases in high level activities that are performed by each role. The activities provide information on what to do, but not how to do it. Scrum is an agile methodology. The people applying Scrum are provided with guidance, but they will require flexibility and experience to manage the project work and complete the activities. Their inspection of the unique project circumstances and characteristics is mandatory to the successful completion of any activity.

Scrum is not a cookbook. It is a guide. A key part of the methodology is training of the three key managers of a Scrum project: the Product Owner (or customer), the ScrumMaster (or project manager), and the development team(s). Training provides these parties with the insights required to intelligently apply Scrum to their project.

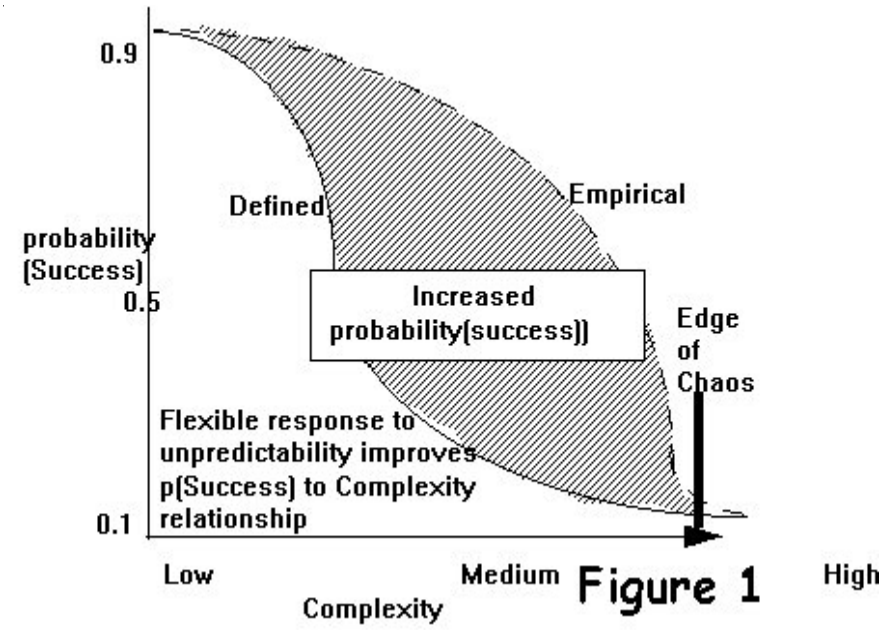


Figure 1

Subject: Introduction

Agile processes represent a new, nontraditional way to build complex products and systems. The AgileAlliance (http://www.agilealliance.org) is a group of industry experts that has developed and employed these processes. It first called this class of development processes “agile” at a meeting in Salt Lake City, Utah in February, 2001. Scrum is one of the original agile processes, originating in the early 1990’s. Other agile processes have emerged since then, including Extreme Programming, Crystal, Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method. Collectively, they have been successfully employed in thousands of projects. The project management activities used in Scrum are based on principles common to all agile processes.

Scrum requires management to learn and use different methods from traditional methods of managing product and system development. All project management requires planning, justifying and initiating the project. Traditional project management duties include establishing and maintaining PERT charts, producing status reports, orchestrating reviews, tracking resources, ensuring time reporting, assigning tasks, and presenting deliverables. By comparison, Scrum project management duties include Sprint and release management, impediment removal, and coaching. Many of the traditional functions of management are unnecessary and irrelevant with Scrum project management.

An Example

I recently helped initiate a project at a large energy company. What follows is a full description of all of the work that was done to initiate the project; through the description, some of the differences between traditional and agile project management become clear.

The customer and I considered formal training to implement Scrum, but we agreed that approach was too academic and probably wouldn’t be real enough to the team. They would have trouble implementing what they learned. We can describe how to ride a bike to someone, but they don’t really “get it” until they experience how to balance while pedaling and moving toward a destination. Agile processes feel different and the teams need to get that feel. If we just taught practices, the team wouldn’t know when and how to apply them without blindly following rules. We thought that if we implanted how agile feels, the practices would fit in and be easy to remember. Accordingly, we orchestrated a workshop to initiate the project.

Prior to the project initiation workshop, the project managers were reluctant to get started. They wanted to discuss things more, think things through more. Think about how you feel before jumping into a cold lake to go swimming for a metaphor. Although they were interested in agile processes, their projects were real life and important to them. They just didn’t see how agile would work. In particular, they didn’t understand how they would be able to ensure that the teams were doing everything that they needed to do.

Entering the workshop, these project managers looked and acted worried. They were used to starting projects by building project plans and PERT charts, and these were absent. To make matters worse, the project teams were new to both project managers, consisting of outside contractors. The project managers looked and acted like they were entering into an arranged marriage with their second cousins.

Subject: Introduction

The workshop flow was:

1. We present agile concepts and overview Scrum and XP.
2. Customer presents business domain.
3. We present product backlog, Sprint Planning, and Sprint concepts.
4. Team members introduce themselves.
5. Customer and team define a product backlog (prioritized requirements list) with enough work to drive the team for several months of Sprints.
6. Customer and team brainstorm about how much functionality the team can build in the next Sprint.
7. Team defines the Sprint backlog (their collective tasks) for the next Sprint to turn the selected product backlog into working functionality.
8. We present daily Scrum, end of Sprint, Sprint signature, and management topics.
9. We present Extreme Programming (another agile process) practices and how they work with Scrum, for Scrum was used to wrap the Extreme Programming engineering practices.
10. The project team starts the first Sprint.

The project managers completely turned around at step #6, when the team defines the work for the next Sprint. At that moment, it was as though a burden had been lifted from each of their shoulders. We start step #6 by reviewing the selected backlog, or what the team is thinking about turning into coded, working functionality by the end of the next Sprint. We then ask the team to spend the next four hours figuring out how they are going to create this working functionality, including a rough design and the work tasks. We then don't say anything else. No one except the team members talk.

At first there was an uncomfortable silence. The team members were thinking, "we're in a training session – they'll tell us what to do next!" Except we didn't, and the project manager had to stay silent also. Soon the pressure of the silence acted on the team and team members started offering some initial observations and questions to each other. This quickly escalates, because software developers are usually very opinionated and have lots of ideas. In the normal, top-down project management process, however, they aren't usually asked; they are told!

The team brainstormed, plotted, schemed, and even planned how it would build the code during the Sprint. The conversations started slowly, but soon people were writing on whiteboards, drawing designs, noodling out approaches and designs for the functionality. As the team discussion progressed, the ScrumMaster wrote down the tasks that were being mentioned. At critical points, the customer or project manager would offer observations, make decisions, and help the team focus on the work. After four hours, we called a timeout. The teams had defined their work for the next Sprint as well as possible. Some unknowns would have to be worked out, some of the estimates were vague, and new work would appear, but the team had gotten a feel for what they would do for the next thirty days.

Subject: Introduction

As the team was brainstorming and planning, a light went off in the project manager's heads. Suddenly they experienced self-organization!! The team would self-organize to figure out what to do. The manager's role was completely shifted to one of facilitator, coach, and mentor. The project manager would help and guide the team to do its best; this is radically different from what they had feared – they were thinking traditionally. They had thought that they would have to define all of the work and ensure that it got done. In Scrum, this is a complete "flip" in responsibilities. The team takes this work from the project manager.

Someone may have read about Scrum, heard about Scrum, and can even talk about Scrum. They use all of the right words. Until they experience Scrum at work, however, it's all academic. When we plan to introduce Scrum into new projects, we now take this into account. We now prioritize giving everyone the actual experience as soon as possible.

Systems development isn't a defined manufacturing activity turning out the same system over and over again. Systems development is a research and development activity wherein teams of professionals wrest product functionality from emergent requirements and new or difficult technologies. Scrum is an empirical process control model that manages and controls such work using frequent inspections and empirical adaptation. The Agile processes and the Agile movement are based on this certainty.

Agile processes are full of deceptively simple practices that have profound ramifications. For example, several agile processes require that developers create a test case prior to writing code for a piece of functionality. Seemingly reasonable and straightforward, this practice requires the developer to have thought through the specification and design of the functionality prior to writing a single line of code – and after the code is written he or she has the test in hand to see if it really works. This simple practice cuts a swath through the burdensome, pretentious and practically unworkable practices of requirements driven testing and artifact driven coding.
Similarly, Scrum has a simple practice called the Daily Scrum; the development team meets daily to review status. Each team member reports what he or she did since yesterday, what he or she will do today, and also to identify anything that's getting in the way. This practice has individuals commit to peers what they will accomplish daily, and then report to their peers whether they were able to carry out their commitment. Deep trust and shared responsibility and commitment result from this daily individual honesty. Team members cover for each other's weaknesses, helping each other accomplish their commitments and self-organizing into otherwise unattainable productivity.

Through these and other equally effective practices - all rooted in empirical process control theory – Agile processes deliver productivity. Not multiples of productivity, but exponential gains. Productivity measured in the rapid, iterative delivery of top priority business functionality. Productivity delivered by only building architectural components and features as they are needed. Productivity measured in business value, not lines of code.

At the root of all Agile processes is the understanding, based on experience and experimentation, that the requirements, architectures, and designs of systems emerge, and that teams of customers and developers self-organize to create the greatest business value iteratively and incrementally. Emergence and self-organization are the threads of agility, rooted in the frequent inspections and adaptation provided by iterative, incremental development.

Subject: *Overview of Agile Processes*

By Martin Fowler, www.martinfowler.com/newMethodology.html. "Most software development is a chaotic activity, often characterized by the phrase "code and fix". The software is written without much of an underlying plan, and the design of the system is cobbled together from many short term decisions. This actually works pretty well as the system is small, but as the system grows it becomes increasingly difficult to add new features to the system. Furthermore bugs become increasingly prevalent and increasingly difficult to fix. A typical sign of such a system is a long test phase after the system is "feature complete". Such a long test phase plays havoc with schedules as testing and debugging is impossible to schedule.

We've lived with this style of development for a long time, but we've also had an alternative for a long time: methodology. Methodologies impose a disciplined process upon software development with the aim of making software development more predictable and more efficient. They do this by developing a detailed process with a strong emphasis on planning inspired by other engineering disciplines.
These methodologies have been around for a long time. They've not been noticeable for being terribly successful. They are even less noted for being popular. The most frequent criticism of these methodologies is that they are bureaucratic. There's so much stuff to do to follow the methodology that the whole pace of development slows down. Hence they are often referred to as heavy methodologies, or to use Jim Highsmith's term: monumental methodologies.

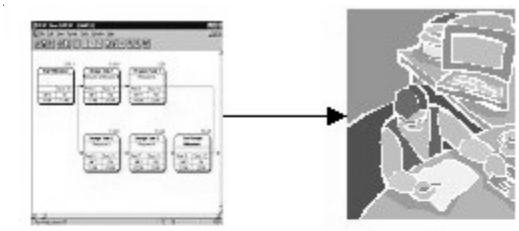
As a reaction to these methodologies, a new group of methodologies have appeared in the last few years. For a while these were known a lightweight methodologies, but now the accepted term is agile methodologies. For many people the appeal of these agile methodologies is their reaction to the bureaucracy of the monumental methodologies. These new methods attempt a useful compromise between no process and too much process, providing just enough process to gain a reasonable payoff.

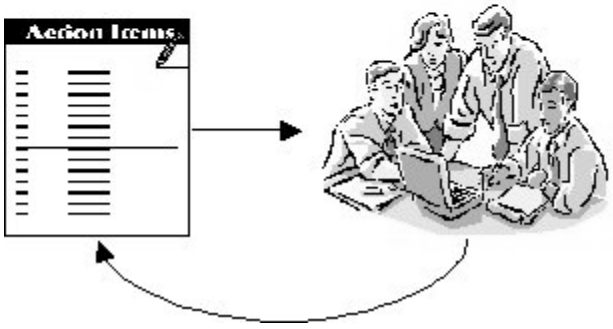
The result of all of this is that agile methods have some significant changes in emphasis from heavyweight methods. The most immediate difference is that they are less document-oriented, usually emphasizing a smaller amount of documentation for a given task. In many ways they are rather code-oriented: following a route that says that the key part of documentation is source code.

However I don't think this is the key point about agile methods. Lack of documentation is a symptom of two much deeper differences:

- *Agile methods are adaptive rather than predictive.* Heavy methods tend to try to plan out a large part of the software process in great detail for a long span of time, this works well until things change. So their nature is to resist change. The agile methods, however, welcome change. They try to be processes that adapt and thrive on change, even to the point of changing themselves.
- *Agile methods are people-oriented rather than process-oriented.* They explicitly make a point of trying to work with peoples' nature rather than against them and to emphasize that software development should be an enjoyable activity. "

Agile Structure





Agile processes use empirical, adaptive processes for rendering emergent requirements into a working system. An illustration of a traditional, defined process shows individuals sequentially completing a list of tasks, each task telling them precisely what to do and how long it will take to do. The illustration is linear, with each task done after the other until all are completed.

An agile process has a skeleton and a heart. The skeleton is the framework on which the rest of the process works. It supports the iterative, incremental nature of all agile processes. Its backbone is the Sprint. While Scrum uses 30-day Sprints, other agile processes use shorter or slightly longer iterations, but they rarely exceed 60 days. Iteration length is limited so that the team is forced to deliver a potentially shippable product increment with regularityt. At the end of every Sprint, the increment is inspected, which dictates the project's future progress. If the increment is appropriate and useful, the project progresses with only minor changes. If the increment is unsatisfactory, the cause is determined and adjustments made before the next Sprint begins. Anything unexpected can be detected and adjusted to at the end of an Sprint, based on the inspection of the increment.

The skeleton is shown in Figure 2. The lower circle represents an Sprint, with the output from each Sprint a potentially shippable product increment. The upper circle represents the daily inspection that occurs during the Sprint, where status is reported and progress inspected and adapted to. This cycle repeats until the project is no longer funded.

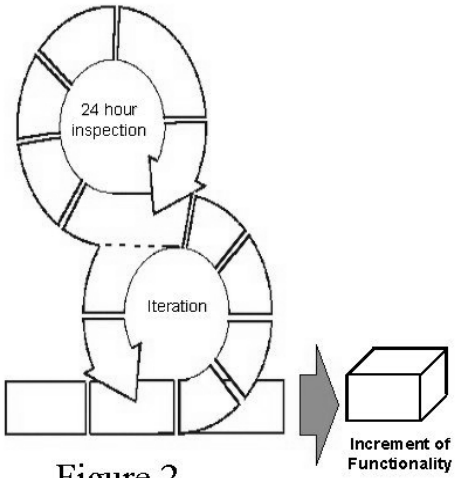


Figure 2

Agile processes try to have all analysis, design, coding and testing work done within an Sprint. Teams perform this work. The optimal team is a small, cross-functional group at the same site (or as "collocated"); but since that's hard to achieve, projects make do and adjust to less-than-optimal circumstances. A new aspect of agile processes introduced by Alistair Cockburn *(Agile Software Development*

Subject: Overview of Agile Processes

Alistair Cockburn, Pearson Education, 2002) measures the impact of less-than-optimal teams, helping management measure the costs and benefits of optimization. Agile processes operate optimally when an organization adopts all of their recommended practices. When scaled to multiple, distributed teams, however, the communication and coordination creates overhead.

The framework operates this way: At the start of an Sprint, the team reviews what it must do. Then, it selects what it believes it can turn into an increment of potentially shippable functionality by the end of the Sprint. The team is then left alone to make its best effort for the rest of the Sprint. At the end of the Sprint, the team presents the increment of functionality that it built.

[1] The heart of agile processes occurs within the Sprint. The team takes a look at the requirements, the technology, and evaluates each other's skills and capabilities. It then devises the best way it knows to build the functionality, modifying the approach daily as it encounters new complexities, difficulties, and surprises. The team figures out what needs to be done, and determines the best way to do it. This creative process is the heart of the extreme productivity that's found in agile processes.
In more traditional project management processes, management devises plans to which teams adhere. The creative process is largely performed outside the team, as management tries to predict the best way for the team to build functionality. There is very little room left for team creativity and flexibility.

I've implemented agile processes many times in projects that are stuck in so many preliminary activities that the teams can't get around to building user functionality. These activities include setting up development environments, selecting the implementation hardware or software, figuring out how to conform to external standards (such as FDA or DOD), or laying out detailed systems architectures. Although this work is necessary, it has these problems:

1. The customer doesn't value it. Although the development organization may know it's required, the customer views it as overhead;
2. This preliminary work is usually performed by specialized groups that aren't responsible for delivering the increments of functionality to the customers; and,
3. It delays the delivery of valuable functionality and can allow the competition to catch up or gain an advantage.

To avoid these problems, development teams do all this preliminary work in agile processes while they're building functionality. If the development teams require experts to do the work, they're included on the teams for as many iterations as it takes. However, this preliminary work is done in parallel with the development of working functionality that can be demonstrated to the customer
In the early iterations, more time is usually allocated to preliminary work. Later iterations mostly develop customer functionality. But in the end, every Sprint must demonstrate working user functionality. Andy Hunt and Dave Thomas (*The Pragmatic Programmer*, Addison Wesley Longman, 2000, Andy Hunt and David Thomas) describe a technique for accomplishing functionality in early iterations called "tracer bullets." Here's how it works: A single piece of functionality is developed within the overall system structure and the development environment. This functionality works from the user interface, through all intermediate levels to the persistent data stores and back. This "tracer bullet" demonstrates that the development environment and operational environment work, allowing the team and users to refine their aim in future iterations.

Subject: *Overview of the Scrum Process*

Scrum is a process for developing products such as software and quickly getting the products to the customer. Scrum is based on empirical controls through inspection and adaptation. These controls are implemented through five basic practices:

1. Iterations – All work is done in short iterations, usually lasting thirty days. Inspections occur at the end of every Sprint. The development project cannot proceed in the wrong direction, or in no direction, for more than one Sprint.
2. Increments – An increment of working functionality is produced every Sprint. At the end of the Sprint, this increment is inspected. Artifacts and abstractions are not inspected, only the reality of the results of the Sprint.
3. Emergence – Complex systems emerge unpredictably across time. Their end-state can be anticipated, but cannot be predicted. It is useless to try to predict their end-states, and any such predictions can be dangerously misleading. Scrum de-emphasizes traditional definitions of the requirements, architecture and design of the system. These factors are allowed to emerge across time and have successfully done so on thousands of agile projects.
4. Self-organization – There are many unpredictable factors in software development, ranging from technology to personnel. Given this unpredictability, it is important that management and teams be are given full authority to plan and organize their work as they go, using their intelligence and creativity to deal with the unexpected. They rely on their experience. They also use any of the documented, defined development approaches (e.g. object modeling techniques, PERT charting techniques, use case requirements capturing) that they deem helpful.
5. Collaboration – The prior control, “self-organization” works only if everyone collaborates freely and openly with each other, contributing based on their capabilities instead of their roles. The practices and working environment of agile processes facilitate collaboration, such as the open working environments and paired programming practices.

Subject: Overview of the Scrum Process

A person trained in Scrum, called the **ScrumMaster**, is responsible for setting up all of the Scrum meetings and ensuring that all Scrum participants follow Scrum's practices and rules.

Scrum consists of iterations called **Sprints**. Each **Sprint** is thirty calendar days in length. During the **Sprint**, a **development team (**or **team)** builds an **Increment** of potentially shippable product functionality. To initiate a **Sprint**, a **Product Owner** meets with the Team for a 1-day **Sprint Planning Meeting**. At this meeting, the **Product Owner** reviews the top priority requirements on a **Product Backlog**, which is a prioritized list of all requirements that are known for the product or system at that time. The **development team** selects that top priority **Product Backlog** that it believes it can turn into an **increment of potentially shippable product functionality (increment**) within the next **Sprint**. Once the **Product Owner** and the **development team** have agreed on what **Product Backlog** to select **(team** selects the amount of top priority Product Backlog), the **team** develops a list of tasks needed to build the functionality. This list of tasks emerges throughout the **Sprint** and is called a **Sprint Backlog**.

The **Sprint** starts immediately after the **Sprint Planning Meeting**. Every day, the **ScrumMaster** meets with the **development team** for a short daily status meeting called the **Daily Scrum**. At the end of the **Sprint**, the **development team**, **ScrumMaster**, **Product Owner**, and other stakeholders (customers, users) meet at a **Sprint Review Meeting**. At this meeting, the **development team** demonstrates the increment of functionality that it developed during the **Sprint**.

The Scrum process consists of the following:

Sprint – a thirty-day iteration of work that results in an increment of product functionality;

Daily Scrum – a standup meeting where status is exchanged, progress is observed, and impediments noted and removed;

Product Backlog – an emerging, prioritized list of user requirements that originated in the product definition;

Sprint Planning Meeting - where each Sprint is planned;

Sprint Review Meeting - where the results of the Sprint are reviewed;

Sprint Backlog – a list of tasks that the team completes to turn product backlog into a product increment during a Sprint;

Customer, user, stakeholder - anyone with an interest in the product or system being constructed;

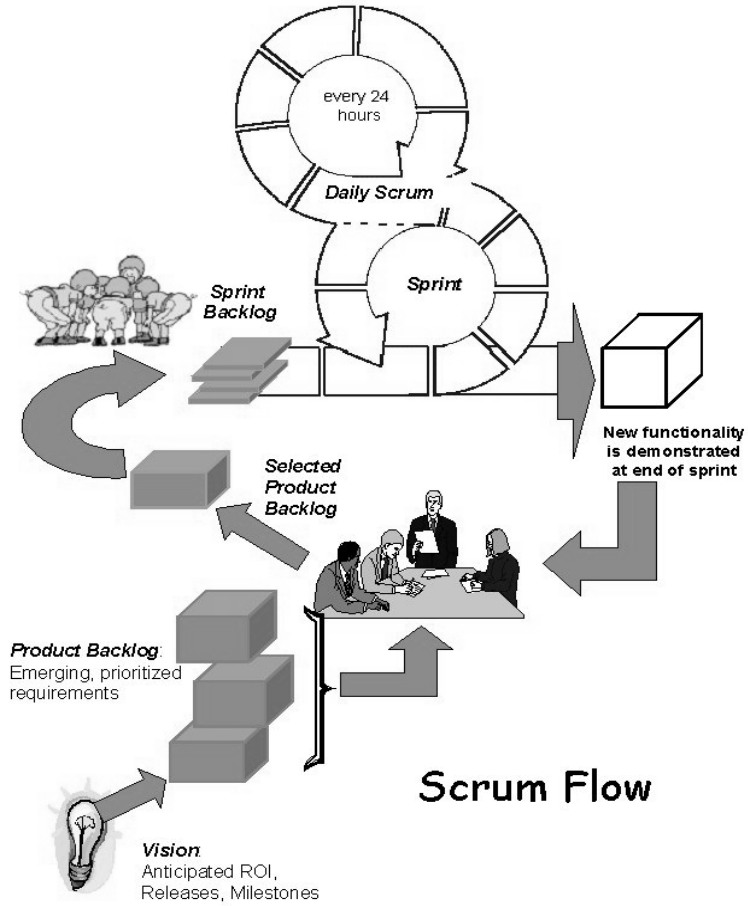
Product Owner - the person responsible for maximizing the value of the product to the customers, users and stakeholders;

Development Team - one or more small teams that develop the system or product;

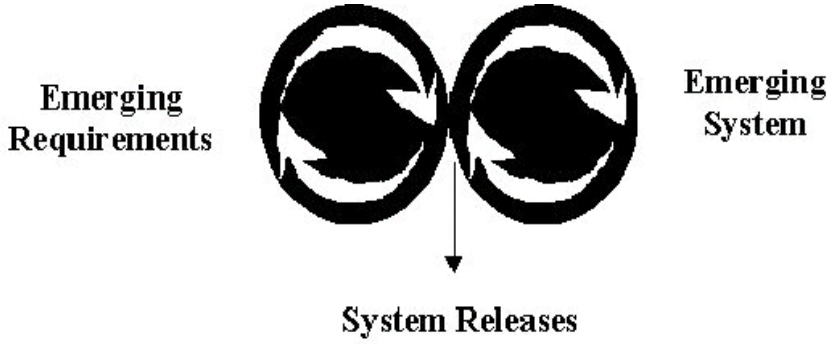
ScrumMaster - the coach of the teams that is responsible for the Scrum process and the productivity of the teams; and,

Increment - an increment of potentially shippable product functionality that a team builds every Sprint.

Subject: Overview of the Scrum Process



Scrum uses two complementary, parallel cycles to build releases of a system. One cycle sustains an emerging list of prioritized requirements called the product backlog. The other cycle consists of development iterations that build system increments of these requirements. These cycles are constructed so that the most appropriate and most essential system emerges over the project's iterations.



These cycles are implemented through the following steps:

1. A Scrum project starts with a vision of the system. The vision may be vague at first, stated in market terms rather than system terms. The vision will become clearer as the project moves forward. A metaphor of the system may also be defined to help guide development and to provide a tangible communication model between users and developers. An initial vision and metaphor can be usually created in a shortened Sprint.
2. The Product Owner, ScrumMaster and development team define and prioritize an initial product backlog focusing on short-term requirements and releases that will extract the most value from the vision.
3. The development team works during the Sprint to create an executable increment that contains the top priority requirements. The team selects as many requirements as it can build during the Sprint. They only build the architecture and design needed to deliver functionality for these requirements.
4. The Product Owner continues defining requirements that will deliver value. They are added to the prioritized Product Backlog. The Product Backlog changes during the project as the business conditions change and as users respond to product increments.
5. At the end of every Sprint, the Product Owner (and customers, users, and stakeholders) review the working system increment with the development team to see if it delivers the expected value, and – if not – what changes need to be made. These changes are added to and prioritized within the Product Backlog.
6. When the Product Owner wants to realize the value achieved to date, he or she can request that product increments built to date be released. One or more Sprints will be used to polish and implement the system into a releasable product.

Scrum Management

When a business operation decides that it needs a system or new product, it assigns someone to head up the effort. For internal systems, this is often a department head. For product companies, this is often a product manager. Scrum refers to this person as the Product Owner. To help formulate and execute the project, the Product Owner turns to the IT or engineering organization for support and staffing. The first step for IT is to assign an IT project manager to work with the product owner. The IT project manager, known in Scrum as the ScrumMaster, becomes a coach, forming teams, coaching and mentoring the individuals on the teams, and helping the teams optimize their productivity. The ScrumMaster and the Product Owner collaborate to maximize ROI and control deliverables at the macro level while helping the team control and organize itself at the micro level.

Subject: Overview of the Scrum Process

The Product Owner develops a plan, a roadmap of the future. Only the most immediate elements of the plan are detailed, however. The rest remains a vision forecast through broad functionality and release goal descriptions. Even this vision is broadly adjusted on an ongoing basis throughout the project. The Product Owner exerts control and risk reduction empirically, responding to project realities rather than trying to make the realities fit a plan. Projects are run iteratively, with increments of functionality delivered every Sprint. The Product Owner manages the progress, the results, and the interaction of business conditions, technology, and people daily, and in detail at the end of every Sprint. The Product Owner plans for each new Sprint based on the empirical evidence of what can be done and what has been done.

Control through inspection and adaptation is also uncommon in IT projects. Scrum provides detailed information and tips on how to empirically manage a project, using agility to maximize the value delivered by the project. Management shifts to assessment, prediction, and coaching. Figure 1 indicates some of the pressure points for exerting this management influence. Every one of these pressure points is used to improve the productivity of the team and the value of the resultant functionality. Agile project management measures productivity in ROI, ignoring the more arcane measures of lines of code and function points. These practices help you stay on top of an effort to produce a valuable system in a complex situation.

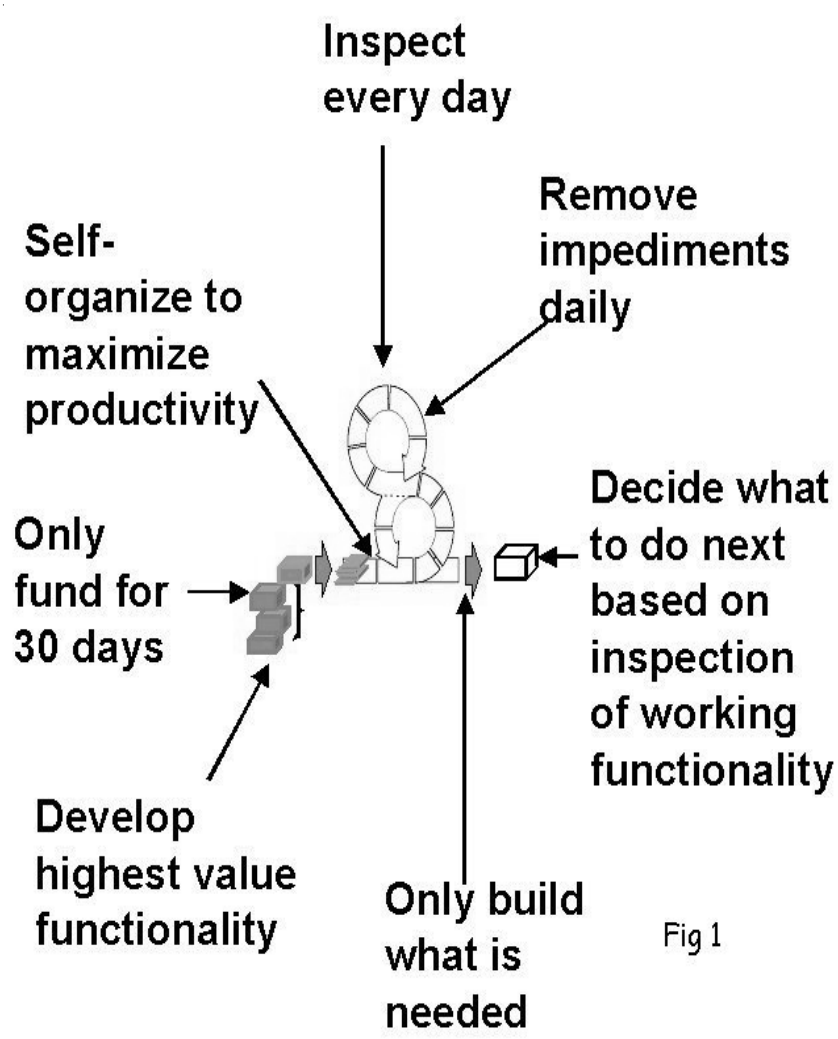


Fig 1

Subject: Overview of the Scrum Process

Scrum project management establishes a vision and general timetables and release goals. Development is conducted in thirty-day iterations. At the end of every thirty days, the Product Owner inspects the results, assesses changes in the business environment, and empirically determines what to do next. The underlying assumption is that the project's events are too detailed to predict. Scrum instead sets up iterations of work and manages the results. Scrum management's consistently zeroes in on the next most valuable thing to do, and maximizes the productivity of the teams doing the work.

Planning as a basis for inspection and empirical response is uncommon in almost all Information Technology (IT) projects. Over the last ten years, this type of planning has been widely adopted in product organizations, however. I've provided details and tips for introducing agile planning practices into the traditional organization found in most IT shops. One practice is the formalization of the "user" role into a Product Owner that is responsible for project ROI; the details of this practice solve the long standing problem of "user involvement" in automation projects. One tip is to initiate projects using three-day workshops. The workshop not only starts the team building product functionality, but it also familiarizes everyone with the new roles and feel of Scrum project management.

When Scrum is used, it is the business unit – that is, the group of people who are going to be using the product in the end – that drives and runs the project from its beginning to its end. In this methodology, the activities that must be performed by the business unit are laid out.

Scrum occurs proactively and intelligently. The Product Owner and ScrumMaster don't study reports; they go out and inspect project activities. The expectation that project management can be applied by rote is abandoned prior to entering the realm of Scrum.

The benefits of Scrum include:

1. Putting the Product Owner (customer, user) in charge of the development project;
2. Allowing the Product Owner to change and create requirements as the project progresses;
3. Ensuring that the most important functionality is built first;
4. Always focusing on only the most important functionality;
5. Ensuring that only functionality that the Product Owner wants is built, so that there is nothing extra to build, pay for, or maintain;
6. Achieving new working functionality every thirty days;
7. Allowing the Product Owner the choice to release the already built functionality at any time.
8. Eliminating the need for projects to be funded more than thirty days in advance;
9. Maximizing the return on investment from the project; and
10. Identifying something can be done to improve productivity every single day.

Subject: Overview of the Scrum Process

Neither the value of Scrum nor the difficulty of implementing it organizationally should be underestimated. Decades and billions of dollars have been spent teaching traditional project management; whole institutions exist to teach and certify individuals. Almost everyone in IT has either been learned and employed this type of project management, or been in projects where they have been subjected to it. The IT culture reflects this, in personnel practices, expectations about what is valuable, and ways of interacting with users. Conversely, users have learned to work with IT within this type of project management, where projects are passed "over the wall" to IT. Over the years, we've learned how to deflect the blame of project failures. Scrum changes all of this.

I've found it very simple and straightforward to resuscitate single projects using Scrum. The results are gratifying; IT gets a taste of agility and the organization gets a taste of success. I've found organization-wide implementation to be much harder. Careful planning, implementation, evaluation, and change management practices are required for agile project management to stick. The results that I've seen at more than one early-adopter show the effort is worth making.

Value driven development

When a systems development project is funded, the Product Owner has a vision of what value the system will bring to the business unit. When planning the project, the Product Owner has to plot out the costs, functionality, implementation dates, and quality. However, these are just planning mechanisms. Too often, however, the Product Owner spends the rest of the project trying to constrain costs, justify slipped dates, and ensure adequate functionality. What value the project will deliver is often forgotten in the heat of the project.

Scrum project management refocuses the Product Owner on the value the system delivers. The value is expressed as a function of the Product Owner's ongoing choice of cost, quality, time, and functionality:

value = f(cost, time, functionality, quality)

Value driven projects leave the determination of the variables in the Product Owner's hands throughout the project. The Product Owner authorizes development Sprint by Sprint, and is free to change any of the variables based on progress to date and delivered value. For instance, if deregulation occurs, the requirements for an energy company project may significantly change. The Product Owner may want to increase the cost and bring in the date to maximize value.

At the end of a Sprint, the Product Owner reviews the working system functionality with the team. Based on the review, the Product Owner can:

- Reprioritize and change the next set of top priority requirements;
- Request that the demonstrated increment be implemented;
- Increase the cost of future Sprints by requesting additional teams to work on the product backlog;
- Adjust the quality to increase or decrease the amount of functionality delivered in an Sprint; and
- Not fund additional Sprints because the value received for the cost is inadequate.

Subject: *Overview of the Scrum Methodology*

Scrum Management Roles

Project management is critical to the success of projects, even projects following agile processes. Without management, project teams may pursue the wrong project, may not include the right mix of personalities or skills, may be impeded by organizational dysfunctionality, or may not deliver as much value as possible. The table below outlines the management roles and responsibilities within a Scrum project. The Scrum methodology consists of various activities that are assigned to these roles.

Activity	Owner	Responsibilities
Manage the vision	Product Owner	The Product Owner establishes, nurtures and communicates the product vision. He achieves initial and on-going funding for the project by creating initial release plans and the initial Product Backlog.
Manage the ROI	Product Owner	The Product Owner monitors the project against its ROI goals and an investment vision. He updates and prioritizes the Product Backlog to ensure that the most valuable functionality is produced first and built upon. He prioritizes and refines the Product Backlog and measures success against expenses.
Manage the development iteration	Team	During an iteration the team selects and develops the highest-priority features on the Product Backlog. Collectively, the team expands Product Backlog items into more explicit tasks on a Sprint Backlog and then manages its own work and self-organizes around how it desires to complete the iteration. The team manages itself to its commitments.
Manage the process	Scrum Master	The Scrum Master is responsible for setting the team up for success by ensuring the project and organizational culture are optimized for meeting the ROI goals of the project. This involves organizing a Sprint Planning Meeting (during which the team expands Product Backlog into Sprint Backlog), a Sprint Review Meeting (during which the newly developed functionality is demonstrated), shielding the team from outside disturbances, holding brief Daily Scrum meetings, and removing obstacles to progress.
Manage the release	Product Owner	The Product Owner makes decisions about when to create an official release. For a variety of reasons it may not be desirable to release at the conclusion of every increment. Similarly, if an official release is planned for after the fifth increment it may be released (with fewer features) after the fourth increment in order to respond to competitive moves or capture early market share. The Product Owner makes these decisions in a manner consistent with the investment vision that has been established for the project.

Meetings

There are three primary meetings in Scrum, the meeting that initiates a Sprint, a daily status meeting during the Sprint, and the meeting that ends the Sprint. These meetings largely replace other formal and management meetings held during development projects. These meetings are thoroughly described in the methodology Activities. Below are summaries:

Sprint Planning Meeting - the Sprint is planned during this meeting. The meeting consist of two parts, each usually lasting four hours:

- Backlog selection - the Product Owner presents the highest priority backlog to the development team. They collaborate about how much can be turned into an increment of potentially shippable product functionality during the next Sprint. The team selects as much as they believe they can handle.

- Sprint workload planning - the team defines the architecture and design of the functionality that it has selected, and then defines the work, or tasks, to build that functionality during the next thirty calendar days.

Daily Scrum - is a short daily status meeting (usually no longer than fifteen minutes) for the development team.

Sprint Review Meeting - is an informal presentation by the team of what the functionality it has developed during the Sprint to the Product Owner and "stakeholders" in the project. The meeting usually lasts four hours. A project retrospective is held at the end of the meeting, where ways to improve the next Sprint are explored and implemented.

Artifacts

Product Backlog, Sprint Backlog, Increment of Potentially Shippable Product Functionality

Product Backlog

An evolving, prioritized queue of functionality is called the Product Backlog.

The requirements for the system or product being developed by the project(s) are listed in the Product Backlog. The Product Owner is responsible for the Product Backlog, its contents, its availability, and its prioritization. The Product Backlog represents everything that anyone interested in the product or process has thought is needed or would be a good idea in the product. It is a list of all features, functions, technologies, enhancements, and bug fixes that constitute the changes that will be made to the product for future releases. Anything that represents work to be done on the product is included in Product Backlog. These are examples of items that would go on the Product Backlog:

- Allow users to access and view account balances for last six months;
- Support distributed development teams;
- Improve scalability of product;
- Simplify installation process when multiple databases are used; and,
- Determine how workflow can be added to product.

Product Backlog isnever complete, and the initial cut at developing it only lays out the initially known and well understood requirements. Sources of Product Backlog are as formal or informal as the hosting organization. The first Product Backlog may be a list of requirements that is gleaned from a vision document, garnered from a brainstorming session, or derived from a marketing requirements document. To get the first Sprint going, Product Backlog only needs to contain enough requirements to drive a thirty-day Sprint. A Sprint can start from only concepts and a wish list.

The Product Backlog evolves as the product and the environment in which it will be used evolves. Backlog is dynamic, in that management constantly changes it to identify what the product needs to be appropriate, competitive, and useful. As long as a product exists, Product Backlog also exists.

Backlog originates from many sources. Product marketing will generate features and functions. Sales will generate backlog that will cause the product to be more competitive or please a particular customer. Engineering introduces backlog that builds technology that holds the whole product together. Customer Support enters backlog to fix major product flaws.
Product Backlog is sorted in order of priority. Top priority Product Backlog drives immediate development activities. The higher a backlog's priority, the more urgent it is, the more it has been thought about, and the more consensus there is regarding its value. The higher the priority, the clearer and more detailed the backlog. Better estimates are made based on the greater clarity and increased

detail. The lower the priority, the less the detail, until you can barely make out the backlog item. In addition to product features and technology, backlog items include issues. Issues require resolution before one or more backlog items can be worked on. For example, if response time is erratic and becoming and hot topic in the industry press, then this might be included as an issue in the Backlog. This issue is not ready to be defined as something to develop into a product. However, it needs to be dealt with and perhaps turned into Product Backlog in the form of features or technology to be developed. Issues are prioritized, just like regular Product Backlog. The Product Owner is responsible for turning issues into work that the Scrum Team selects for a Sprint. Until he or she converts the issue to regular Product Backlog, it remains as "unworkable" Product Backlog. This ensures that the team isn't swamped by having to think about outstanding issues while it works.

As a product is used, as its value increases, and as the marketplace provides feedback, the product's backlog becomes larger and more comprehensive. What a team needs to do never stops changing, and so the requirements never stop changing. It makes little sense to pretend that this is not the case and attempt to set requirements in stone before beginning design and construction.

All you need in Scrum is a product vision and enough top priority items on the backlog to begin one Sprint, or Sprint, of incremental development on the product. The rest emerges. Product Backlog is an inventory item for software development, and it is never accumulated more than necessary.

An example of Product Backlog maintained on the Scrum Product Management tool, based in a spreadsheet, looks like:

Subject: Overview of the Scrum Methodology

This spreadsheet is the Product Backlog in March, 2003, for the Scrum Project Management software. The rows are the backlog items, interspersed by Sprint and Release dividers. For instance,

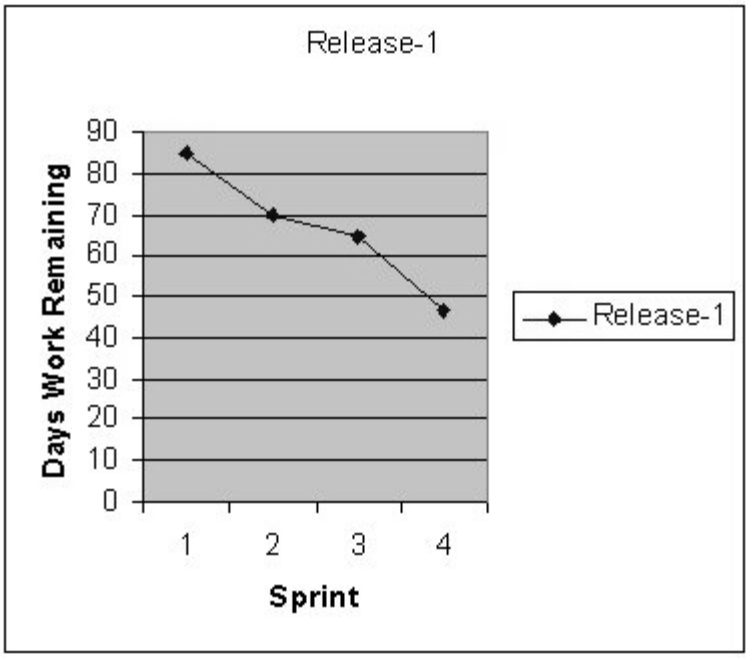
Backlog Description	Initial Estimate	Adjustment Factor	Adjusted Estimate	Remaining work remaining until completion						
				1	2	3	4	5	6	7
Title Import				256	209	193	140	140	140	140
Project selection or new	3	0.2	3.6	3.6	0	0	0	0	0	0
Template backlog for new projects	2	0.2	2.4	2.4	0	0	0	0	0	0
Create product backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Create sprint backlog worksheet with formatting	3	0.2	3.6	3.6	0	0	0	0	0	0
Display tree view of product backlog, releases, sprints	2	0.2	2.4	2.4	0	0	0	0	0	0
Sprint-1	13	0.2	15.6	16	0	0	0	0	0	0
Create a new window containing product backlog template	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Create a new window containing sprint backlog template	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Burndown window of product backlog	5	0.2	6	6	6	0	0	0	0	0
Burndown window of sprint backlog	1	0.2	1.2	1.2	1.2	0	0	0	0	0
Display tree view of product backlog, releases, sprints	2	0.2	2.4	2.4	2.4	0	0	0	0	0
Display burndown for selected sprint or release	3	0.2	3.6	3.6	3.6	0	0	0	0	0
Sprint-2	16	0.2	19.2	19	19	1.2	0	0	0	0
Automatic recalculating of values and totals	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
As changes are made to backlog in secondary window, update burndown graph on main page	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Hide/automatic redisplay of burndown window	3	0.2	3.6	3.6	3.6	3.6	0	0	0	0
Insert Sprint capability ... adds summing Sprint row	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Insert Release capability adds summary row for backlog in Sprint	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Owner/assigned capability and columns optional	2	0.2	2.4	2.4	2.4	2.4	0	0	0	0
Print burndown graphs	1	0.2	1.2	1.2	1.2	1.2	0	0	0	0
Sprint-3	14	0.2	16.8	17	17	17	0	0	0	0
Duplicate incomplete backlog without affecting totals	5	0.2	6	6	6	6	6	6	6	6
Note capability	6	0.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2	7.2
What-if release capability on burndown graph	15	0.2	18	18	18	18	18	18	18	18
Trend capability on burndown window	2	0.2	2.4	2.4	2.4	2.4	2.4	2.4	2.4	2.4
Publish facility for entire project, publishing it as HTML web pages	11	0.2	13.2	0	0	13	13	13	13	13
Future Sprints	39	0.2	46.8	34	34	47	47	47	47	47
Release-1				85	70	65	47	47	47	47

all of the rows above Sprint 1 were worked on in that Sprint. The rows between Sprint 1 and Sprint 2 were done in Sprint 2. Notice that the row “Display tree view of product backlog, releases, sprints” is duplicated in Sprint 1 and Sprint2. This is because row 10 wasn’t completed in Sprint 1, so it was moved down to the Sprint 2 for completion. If we decided that it was lower priority after Sprint 1, we could have moved it even lower down the priority list.

The first four columns are the backlog item name, the initial estimate, the complexity factor, and the adjusted estimate. These are described in more detail later in the methodology. The next columns are the Sprints during which the backlog is developed. When the backlog is first thought of and entered into the Product Backlog, its estimated work is placed into the column of the Sprint that is going on at that time. Most of the backlog items shown were devised by the developers and myself before starting. The sole exception is row 31 (Publish facility for entire project, publishing it as HTML web pages), which we didn’t think of until during Sprint 3.

A burndown chart shows the amount of work remaining across time. If we track this for the anticipated release, it looks like the chart on the right after the first four Sprints. Even though the work declined, the trend line slope decreased after Sprint 2 because the work of the backlog item in row 31 was added to the total work for that Sprint.

The burndown chart is an excellent way of visualizing the impact of working on all of the items in Release 1 while evaluating what should be in Release 1 and adding or removing items. This allows the Product Owner to "what if" by adding and removing functionality from the release to get a more acceptable date. Or extending the date to include more functionality. The burndown chart is the collision of reality (work done and how fast it's being done) with what is planned, or hoped for.



The backlog items in the Product Backlog for Future Sprints is pretty coarse grained. We haven't started any more work on these items, so we haven't expended the time to more finely estimate them. Similarly, there are plenty more requirements for this product. They just haven't been thought through. When we have the time or inclination to start development again, more backlog will be defined. This is a perfect example of the requirements for the product emerging. By deferring building inventory of backlog until we need it, we have been able to get to building functionality without waiting for a detailed analysis of the requirements of what may never be built.

Sprint Backlog

The Sprint Backlog defines the work, or tasks, that a team has defines for turning the Product Backlog it selected for that Sprint into an increment of potentially shippable product functionality. The team consists of everyone who will be doing the actual work. After establishing the Sprint goal, the team determines what work will have to be performed in order to reach the goal. All team members are required to be present when this is determined. The team may also invite other people to attend in order to provide technical or domain advice. The Product Owner often attends, too, but this is the team's meeting. It is often in this meeting that a team realizes that it will either sink or swim as a team, not individually. The team realizes that it must rely on its own ingenuity, creativity, cooperation, collaboration, and effort. As it realizes this, it starts to take on the characteristics and behavior of a real team. During this meeting, management and the user should not do or say anything that takes the team off the hook.

The team compiles a list of tasks it has to complete to meet the Sprint goal. These tasks are the detailed pieces of work needed to convert the Product Backlog into working software. Tasks should have enough detail so that each task takes roughly four to sixteen hours to finish. This task list is called the Sprint Backlog. The team self-organizes to assign and undertake the work in the Sprint Backlog. Sometimes only a partial Sprint Backlog can be created. The team may have to define an initial architecture or create designs before can fully delineate the rest of the tasks. In such a case, the team should define the initial investigation, design, and architecture work in as much detail as possible, and leave reminders for work that will probably have to be done once the investigation or design has been completed. At that time, the work will be more fully understood and can be listed in more detail.

The team modifies its Sprint Backlog throughout the Sprint. As it gets into individual tasks, it may find out that more or fewer tasks are needed, or that a given task will take more or less time than had been expected. As new work becomes required, the team adds it. As tasks are worked on or completed, the hours of estimated remaining work for each task are updated. When tasks are deemed unnecessary, they are removed. Only the team can change its Sprint Backlog during a Sprint. Only the team can change the contents or the estimates. The Sprint Backlog is a highly visible, real time picture of the work that the team plans to accomplish during the Sprint.

Sometimes the Scrum Team discovers that it has selected too much Product Backlog to complete in a single Sprint. If this happens, the ScrumMaster immediately meets with the Product Owner. They jointly identify Product Backlog that can be removed while still meeting the Sprint Goal.
Teams become better at Sprint planning after the third or fourth Sprint. At first, a team tends to be nervous about taking on responsibility and it under-commits. As it becomes more familiar with Scrum processes, as it starts to understand the functionality and technology, and as it gels into a team, it commits to more work.

Task Description	Originator	Responsible	Status (Not Started / In Progress / Completed)	Hours of work remaining unt											
				1	2	3	4	5	6	7	8	9	10	11	12
Meet to discuss the goals and features for Sprint 3-6	Danielle	Danielle/Sue	Completed	20	0	0	0	0	0	0	0	0	0	0	0
Move Calculations out of Crystal Reports	Jim	Allen	Not Started	8	8	8	8	8	8	8	8	8	8	8	8
Get KEG Data		Tom	Completed	12	0	0	0	0	0	0	0	0	0	0	0
Analyse KEG Data - Title		George	In Progress	24	24	24	24	12	10	10	10	10	10	10	10
Analyse KEG Data - Parcel		Tim	Completed	12	12	12	12	12	4	4	4	4	0	0	0
Analyse KEG Data - Encumbrance		Josh	InProgress							12	10	10	10	10	10
Analyse KEG Data - Contact		Danielle	In Progress	24	24	24	24	12	10	8	6	6	6	6	6
Analyse KEG Data - Facilities		Allen	In Progress	24	24	24	24	12	10	10	10	10	10	10	10
Define & build Database		Barry/Dave	In Progress	80	80	80	80	80	80	60	60	60	60	60	60
Validate the size of the KEG database		Tim	Not Started												
Look at KEG Data on the G:\		Dave	In Progress	3	3	3	3	3	3	3	3	3	3	3	3
Confirm Agreement with KEG		Sue	Not Started												
Confirm KEG Staff Availability		Tom	Not Started	1	1	1	1	1	1	1	1	1	1	1	1
Switch JDK to 1.3.1. Run all tests.		Allen	Not Started	8	8	8	8	8	8	8	8	8	8	8	8
Store PDF files in a structure		Jacquie	Completed	8	0	0	0	0	0	0	0	0	0	0	0
TopLink. Cannot get rid of netscape parser		Richard	Completed	4	0	0	0	0	0	0	0	0	0	0	0
Build test data repository		Barry	In Progress	10	10	10	10	10	10	10	10	10	8	8	8
Move application and database to Qual (incl Crystal)		Richard	Completed	4	4	4	4	4	4	4	4	0	0	0	0
Set up Crystal envrionment		Josh	Completed	2	2	2	2	1	1	1	1	0	0	0	0
Test App in Qual		Sue	In Progress												20
Defining sprint goal required for solution in 2002		Lynne	In Progress	40	40	40	40	40	40	40	40	38	38	38	38
Reference tables for import process		Josh	In Progress												
Build standard import exception process		Josh	InProgress										12	12	10
Handle multiple file imports on same page		Jacquie	Disregarded	20	15	15	15	12	12	12	12	12	9	0	0
Migrate CruiseControl Servlet to IWS 6.0 (landcc_7101) server		Allen	Not Started	4	4	4	4	4	4	4	4	4	4	4	4
Create web server for Qual on PF1D8		Allen	Completed	1	0	0	0	0	0	0	0	0	0	0	0
LTCS Disk		Danielle/George	In Progress	12	12	12	12	8	8	8	8	8	8	8	8
Follow thru with questions about KEG data to Sue/Tom, re: Keg, LTO	Jacquie	Danielle	Completed	10	10	10	10	10	8	8	8	0	0	0	0
Map KEG data to Active Tables - see also #14	Jacquie	Jacquie/Allen	In Progress	50	50	50	50	50	50	50	50	50	50	50	50
Prepare SQL to import from KEG tables to Active Tables	Jacquie	George	In Progress	25	25	25	25	25	25	25	25	25	24	23	22

Increment of Potentially Shippable Product Functionality

A number of people have questioned how Scrum and other agile processes support mission critical, FDA approved, or other varieties of software systems. These questions have arisen because the vanilla version of the most popular agile processes describe the construction of software that is single use; that is, it is used within the organization that develops it, with the development mostly done by the internal Information Technology (IT) organization.

Scrum requires development teams to built an increment of product functionality every Sprint. This increment must be potentially shippable, if the customer desires to implement the functionality. This requires that the increment consist of thoroughly tested code that has been built into an executable, and the user operation of the functionality is documented, either in Help files or user documentation.

If the product increment that is created during the Sprint has more exacting use, the development organization usually defines the additional product requirements as standards or conventions. For example, the Federal Drug Administration (FDA) must approve a product that will be used in life-critical circumstances by in numerous health care settings if the product is to be used in the United States. As part of the approval process, the FDA checks that specific information regarding the product is provided, such as requirements traceability and specific functionality operation. For each increment to be potentially shippable, these additional facets of the product must also be developed – so that each increment of the product is potentially ready for FDA approval.

Similarly, some products require that performance requirements be modeled and the performance demonstrated mathematically, not just through statistical measurement of the actual system. The model with all required rigor is thus an additional part of each iteration's potentially shippable increment.

When the Product Owner chooses to implement a set of developed functionality, Sprints are established to release the product. These Sprints finalize the selected increments of functionality into a shippable release. If the increments are fit for their purpose, these Sprints are short and few. For example, if the release is an FDA approved teleradiology system, the increments of functionality should include all of the deliverables that are required for FDA approval. To the extent that these deliverables haven't been included, they must be created and included in the release sprints.

Scaling agile processes to build any unique, non-single use, system means ensuring that the development team builds potentially shippable increments of functionality. If not already present, the standards, conventions, and practices that a development team must follow have to be developed. These standards, conventions and practices are staged as high priority work. Their development in the initial iterations ensures that subsequent Sprint's work will produce potentially shippable increments of functionality.

The phrase used in agile development for such an iterative, incremental process is "sashimi", a thin slice of a product which contains all aspects of the final product. The word sashimi is related to sushi; one slice of sushi is similar to all other slices.

Quality of the Increment

Many organizations are still in quest of quality, even after the advent of Total Quality Management, CMM, and now Six Sigma. Scrum also addresses the issue of quality. Teams are required to demonstrate potentially shippable product functionality at the end of every Sprint. As I discussed previously, the functionality must contain all of the artifacts appropriate for that type of system. Functionality requiring FDA approval requires artifacts for requirements traceability, for instance.

Implicit in the definition of "potentially shippable" is quality. Quality increments of functionality have at least three dimensions:

- External quality. The product has fitness of purpose to the intended user, including ease of use;

- Internal quality. The product is well designed, well structured, free of internal errors, and maintainable and sustainable; and,

- Appropriateness. Systems intended for FDA approval, mission critical, complex, embedded and other types of systems require artifacts in addition to high quality code.

Scrum calls for cross-functional teams. The expertise to create both external and internal quality in each increment is built into the team. The team is asked to build functionality with enough quality that it is potentially shippable. It has to meet all three of the above quality dimensions. There is no passing the buck, saying to a quality group, "here's the code, see if you can find anything wrong with it!"

Scrum implements the agile practices of inspection and adaptation. For inspection to be of value, the inspectors need to have a pretty good idea what they are inspecting. If the increment of working functionality that is being demonstrated has good user quality (that is, looks good), but has poor internal quality (buggy, bad design, duplicate code), then the increment only looks potentially shippable. In reality, quite a bit more work is required prior to actual shipment.

If the appropriate standards, conventions, and practices are followed and the team implements good external and internal quality, the development and release iterations look like figure 6. Most of the work is done in the development iterations building a quality, fit for use product. The release iterations are few and short, mostly polishing the functionality into releasable status.

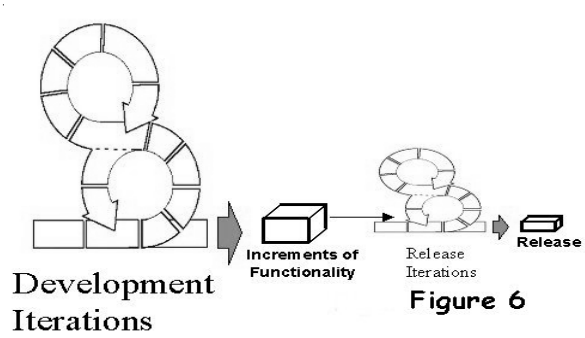


Figure 6

Subject: Overview of the Scrum Methodology

If the appropriate standards, conventions, and practices are **not** staged and developed early - or the team doesn't build in adequate external and internal quality - the development and release iterations look like figure 7. That is, the potentially shippable product increments demonstrated at the end of every Sprint were not suitable for use. They required significant additional work to acquire sufficient quality to ship.

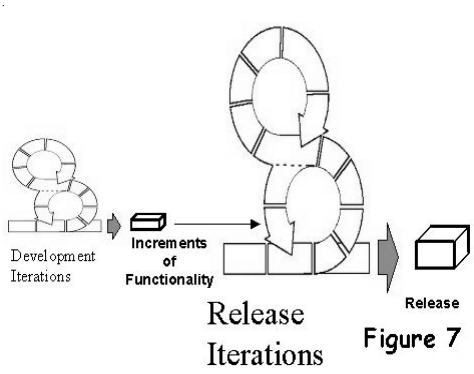


Figure 7

In the circumstances shown in figure 7, the product owner doesn't know what is being inspected. It may look good (external quality), but have inadequate internal or appropriateness quality. Scrum states that the team is supposed to demonstrate potentially shippable functionality, but sometimes-significant additional work is needed prior to shipment. Maybe this has happened because:

- The team doesn't have or follow adequate engineering practices;

- The standards weren't developed for the increment to be usable in its target environment (FDA products, for instance); or,

- Management has pressured the team to develop more functionality in the Sprint than can be developed with adequate quality.

Regardless, the consequences are realized during the release iterations. The product owner probably wants the release soon after the development iterations. However, poor quality makes it difficult if not impossible to assess how many of these iterations will be required. Workload burndown graphs demonstrate what happens to each type of project. A workload burndown graph represents workload on the vertical axis and time on the horizontal axis (divided into 30 day iterations in the figures shown). I track the change in workload across time. When I project the trend from the work burndown, I can estimate when the workload will be complete and the product ready for release. The following two burndown graphs used in Scrum reflect the dilemma posed by not being able to know the quality of the increments of functionality.

In Figure 8. the product functionality has been developed more or less according to schedule and expected quality. The Product Owner had estimated one release Sprint prior to shipment. As the functionality is being readied for shipment or release, there are no surprises and the expected release date is met.

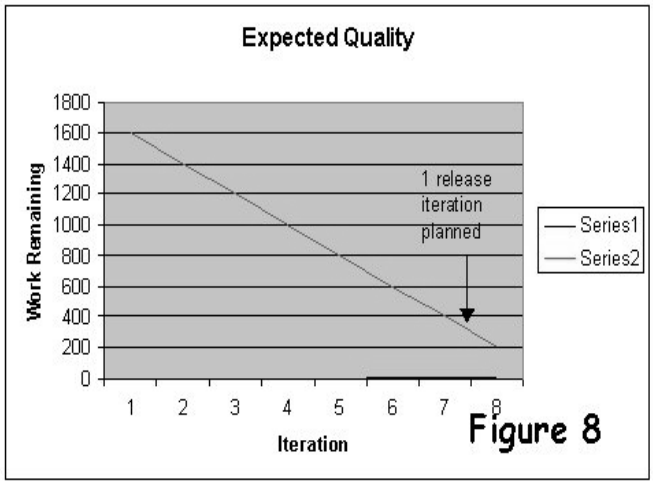


Figure 8

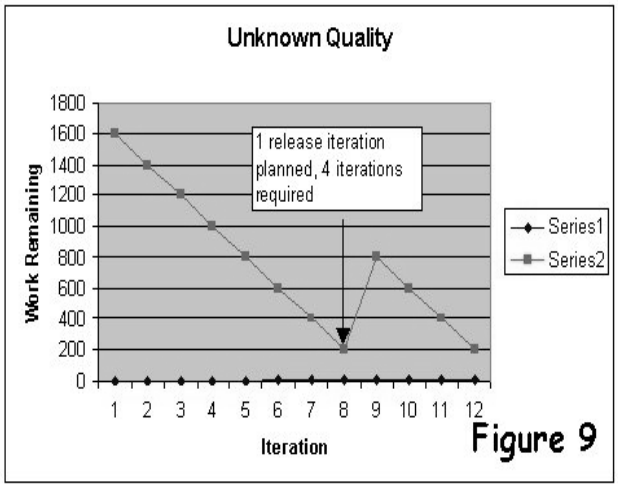


Figure 9

In Figure 9, the product functionality has been developed more or less according to schedule, but the internal quality is poor and the artifacts required to make it releasable to its intended users (FDA, etc.) are not completed. The Product Owner had estimated one release Sprint prior to shipment, but now cannot reliably estimate when this will occur. Figure 9 shows that four additional release iterations were required and that the work is completed linearly (rarely the case). Introducing quality into already developed functionality is unpredictable. One or more of the following scenarios may play out while in the release iterations:

- The internal structure and design is poor. Refactoring has been omitted and duplicate code and structural inefficiencies exist. As the development team attempts to fix one bug, new bugs are introduced. While fixing the new bugs, even more bugs are introduced. Under the pressure of an already blown release date, the team struggles to refactor the design and code while fixing the bugs.

- The functionality is turned over to a quality assurance group for quality testing. They find bugs and report them. The development team can't proceed with new development until these bugs are fixed. However, the Product Owner expected that the team delivered quality functionality and has them working on new functionality. The team is torn between new work and bug fixing.

- To improve quality, I've often had to implement the engineering practice that the development team owns the code forever. I've found that this practice incentivizes everyone to write quality code. The team is motivated to write clean, understandable, maintainable, sustainable code. The team owns the code and either can be proud of it or it will cause them grief forever. Some organizations feel that freeing teams of the responsibility for maintaining the code lets them proceed to new work. In reality, this simply moves fixing the code to people who don't understand it.

When the team owns the code forever, management knows that the team can't be assigned to new functionality iterations until the code is of adequate quality to ship. Management thus resists pressuring the team to build more functionality during iterations than can be built with quality. This simple practice puts both the development team and management on the same page, dedicated to the same goal.

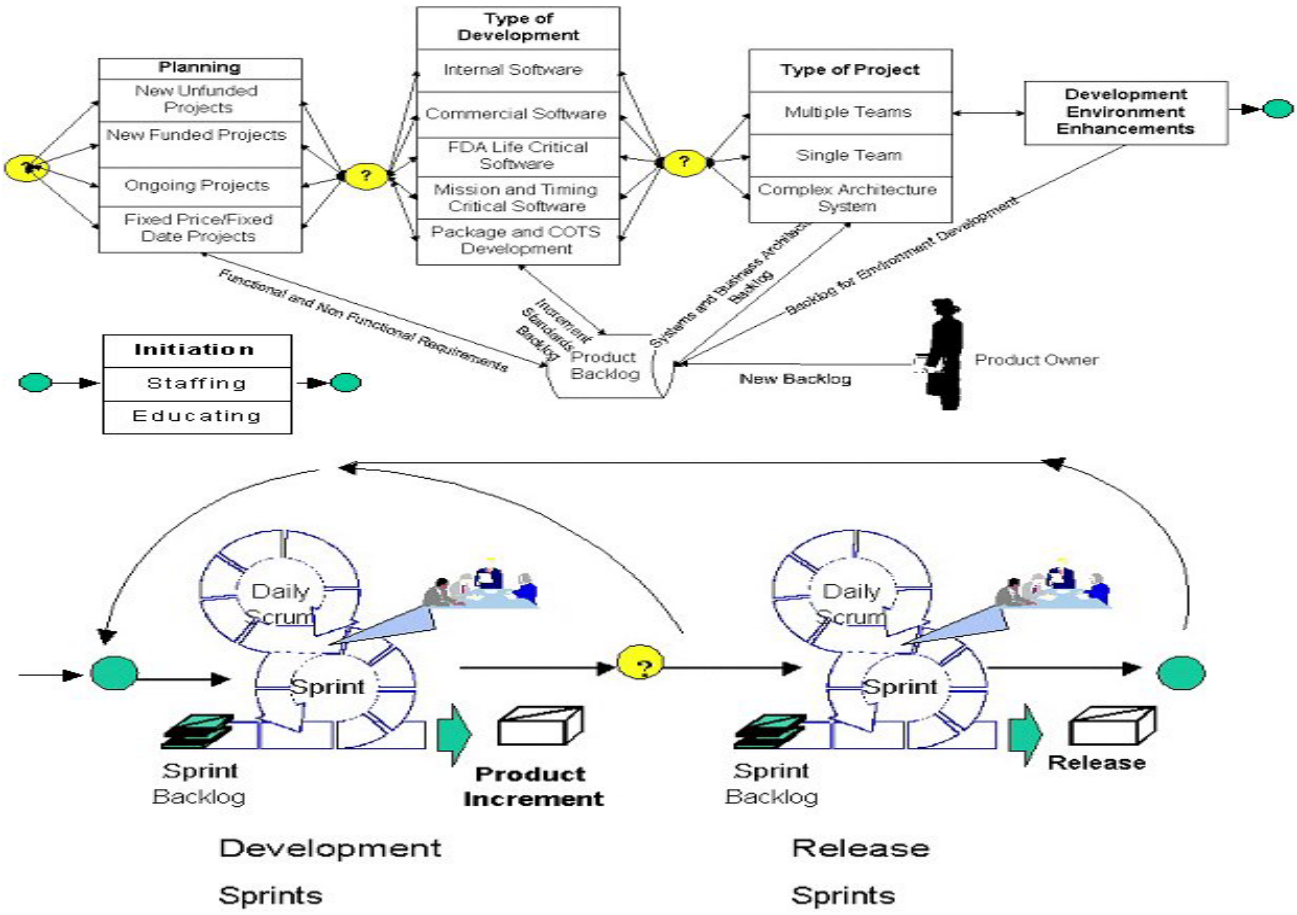
This practice is a strong management statement for quality. The potentially shippable product increment says that the team is authorized to cut functionality width or depth during the Sprint as long as the quality is shippable. We all know that when we pressure developers to put more functionality in by a certain date that we are going to pay for it later. The payment is buggy code that causes customer dissatisfaction, diminished product reputation, and dissatisfied and dispirited developers.

Structure of Phases, Paths, and Activities

The Scrum methodology consists of Phases, Paths, and Activities. Phases are groupings of work that achieves a common purpose. Paths are different ways of structuring the Activities within a Phase based on project characteristics. Activities are the Scrum's work, the stuff that actually constitutes the Scrum methodology. Each Activity has a "Responsible" for party, the Scrum role that is responsible for the successful execution of that activity. For instance, the Product Owner is responsible for initially estimating product backlog (an Activity within the Planning Phase).

Many methodologies use a "task" structure to delineate the lowest level of work, such as "Build A Class Diagram,, "Instantiate a Sequence Diagram," and other detailed pieces of work. Scrum doesn't include tasks. The theoretical underpinnings of Scrum require such work to be determined empirically by those responsible for the work, as the work occurs. For example, a Scrum Development Team is responsible for turning selected Product Backlog into an increment of working product functionality. The actual tasks to do so are derived by the Development Team for its own purposes. The responsible parties figure out how to do the work and then carry it out, modifying the tasks as they work.

Think of Phases as groupings of like Activities with Paths through the Activities depending on the type of project.



Scrum Phases

Planning

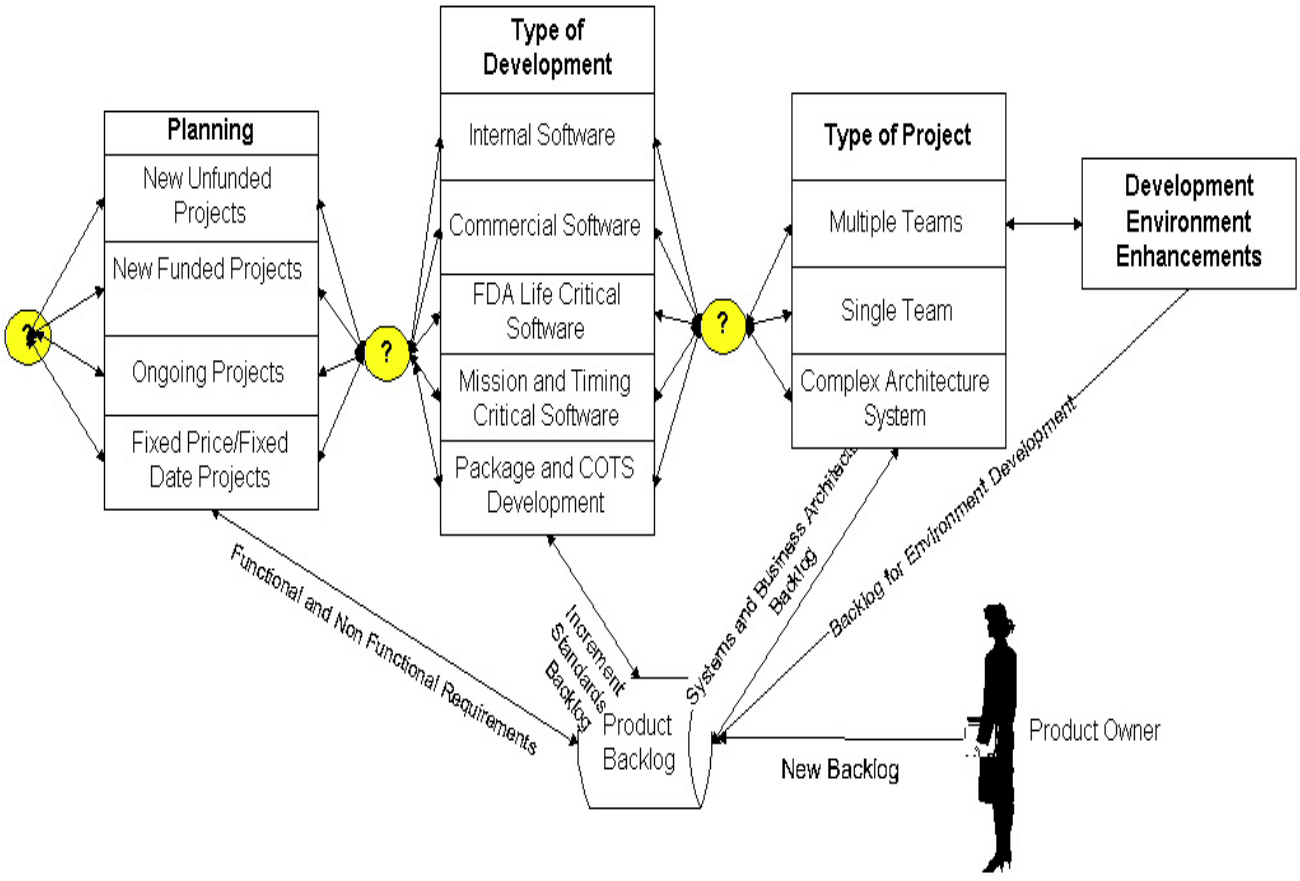
The Planning phase's purpose is to establish the project vision, to establish expectations, and to acquire funding.

The vision states how the business or customer environment will be different when the product being built is successfully installed. The vision may include phasing. For example, one healthcare software vendor envisioned "teleradiology,", a vision based on the introduction on digital imaging equipment such as MRI's and x-Rays. The vision was of a filmless radiology operation where collaborating radiologists worldwide could review, discuss, markup and reach conclusions regarding the same digital image of the patient.

The purpose of the vision is to establish a common context within which decisions can be made. The vision is used to acquire funding by showing what benefits will accrue over time if the vision is successfully implemented, and what the attendant costs will be. A vision consists of words, models, and spreadsheets, and is usually relatively short.

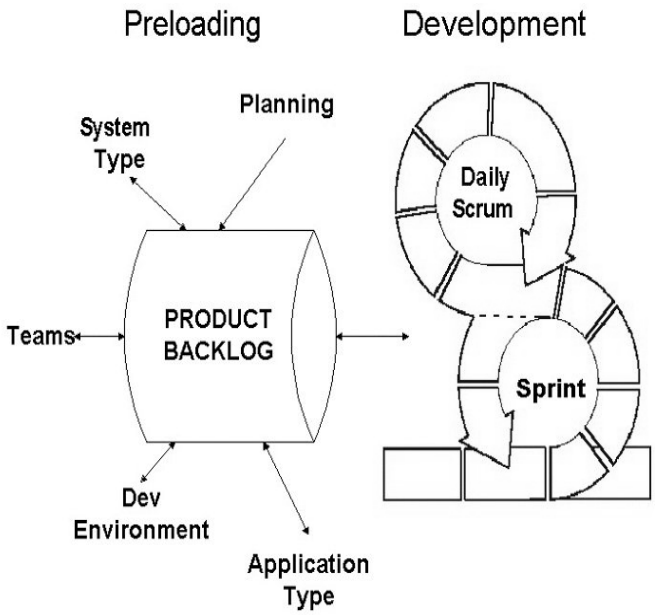
If the expectations set by the vision appeal to sources of funds (capital committees, venture capitalists), funding may be provided to proceed with the project as stated in the vision.

Once Planning is done, funding, an initial Product Backlog and release plan, high level technical and business architectures, and expectations are in hand.



Staging

Projects have different characteristics. Some are large and have many people. Others are small but have several teams working to build very complex software. Some projects require significant work at the beginning to establish a satisfactory development environment. Other projects are ready to go. The Staging phase assesses the various dimensions of the project and creates additional Product Backlog.



There are multiple Phases, or subPhases, in the Staging phase, each representing a single dimension of the project that should be assessed. Based on the assessment, different Paths may be taken through the subPhase. The Activities within each Path don't actually do project work. Instead, the Activities add and prioritize requirements into the Product Backlog. All of the actual work that results from these requirements is done by the Project Team during the Development Phase.

The Staging phase builds communication and coordination mechanisms across the basic unit of Scrum - small, cross-functional, self-organizing teams.

Initiation
Staffing
Educating

The last subphases in the Staging phase constitute the initiation process, or getting the project started. Teams are staffed. The Teams get together with the Product Owner to learn about the project. The Product Owner instills as much vision and project information as possible to the Team. Sometimes this includes sending the team out into the target environment so the team acquires a more detailed understanding of where the system will be implemented. If the Team has never used Scrum before, Scrum process training is provided to the Team.

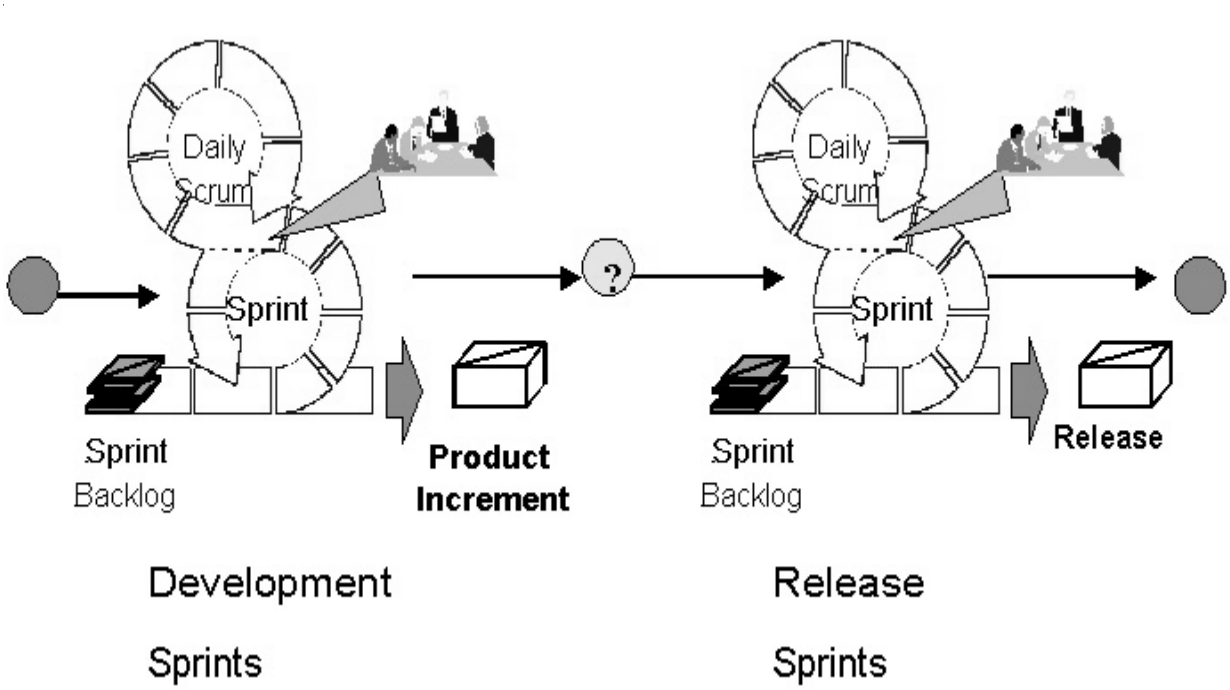
Developing

All actual product development is done in the Developing Phase. This phase consists of multiple Sprints to develop increments of product functionality. Each Sprint starts with a Sprint Planning Meeting and concludes with a Sprint Review.

When projects are building complex, highly architected software or when multiple Teams will be working together on the software, Product Backlog to build coordinating models and architectures are built in the first several Sprints by a single Team. This Team is then distributed to new teams and provides this product contextual information to each Team.

When upgrades to the development environments are required before Teams can successfully develop "potentially shippable" increments of product functionality, this work is scheduled for the first several Sprints by the Product Backlog being given high priority. This environmental work occurs in parallel with development of actual functionality.

Regardless of the architectural or environmental work required in the initial Sprints, every Sprint must deliver an increment of working product functionality.



Releasing

The Vision called for releases of product functionality based on a combination of costs, expected benefits, dates, and available functionality. At the end of every Development Sprint, the Product Owner assesses whether such functionality is available and if the time if right for it to be released. If the Product Owner determines to create a release, the Releasing Phase is entered.

The Releasing Phase consist of two types of Activities. The first Activity adds requirements to the Product Backlog that will turn the "potentially shippable" increments of functionality into a real shippable product. Any shortcomings in the Development Phase Sprints are addressed at this point. The second Activity is one or more Sprints of work that create the releasable product.

Phase Name: *1 Planning*

This section addresses how to fund a wicked problem when you don't know what the solution will be? How can a capital committee allocate funds under these circumstances and be comfortable that the risk is under control and the ROI is predictable. The Product Owner is responsible for planning the project. Within large organizations, this person is often a department head, such as the head of manufacturing or inventory control. Within product organizations, this person may be a product manager, such as for a software product. Sometimes this person is an IT project manager for internal IT infrastructure projects, such as consolidating servers. The Product Owner establishes, nurtures and communicates the product vision. He achieves initial and on-going funding for the project by creating initial release plans and the initial Product Backlog.

Why Plan

Planning is done to establish a vision that is shared between the visionary, the funders, and the people working on the project. This vision binds all of these people together in evaluating the progress of the project, and in making correct decisions within the context of the vision. The plan also makes certain assertions regarding the value of the project and what timetables need to be met to deliver this value. This becomes the benchmark against which the funders and management evaluate the progress of the project.

Planning Basics

Planning the project usually takes one short Sprint of fifteen days to complete. All of the Scrum practices of backlog, Sprints, and daily Scrums apply to this planning Sprint. The output from the planning Sprint is the project definition document and a prototype, or proof of concept, of one part of the functionality running on the technology.

All project planning starts by defining what system is to be developed and why it is important to develop it.

Traditional project planning then develops a detailed picture of all of the functionality and technology that is required to deliver the envisioned system. A plan defining all of the activities and tasks that are required to build the system functionality and technology is then prepared. These tasks are then staffed and scheduled. The resultant plan is used to control and manage the project. Management causes people to do the planned work.

Scrum project management relies on the agile practices of emergence, self-organization, and iterations. The system is being built in a complex business environment on complex technology. Although the system may be envisioned when planning the project, the functionality and technology that actually deliver the system will often change during the course of the project. As new business opportunities arise, the functionality will change. As the users see the functionality, they will change how the functionality should be delivered. As new technology appears or selected technology proves not to be viable, technology will change.

Scrum project management only defines the functionality of the system at a high level. Only functionality that will be built first is detailed, and only to such a level as to make rough estimates. The detailed functionality is the highest priority to build, usually that which delivers the most value to the business. These details are rarely defined for more than the first release, rarely more than six months from project inception.

Phase Name: *1 Planning*

Scrum project management doesn't plan and define work for teams. Instead, the teams define the work for each Sprint at the beginning of each Sprint. The teams self-organize themselves and the work, and then manage themselves in the performance of the work. There is no detailed work planning in a project plan, only a list of expected functionality.

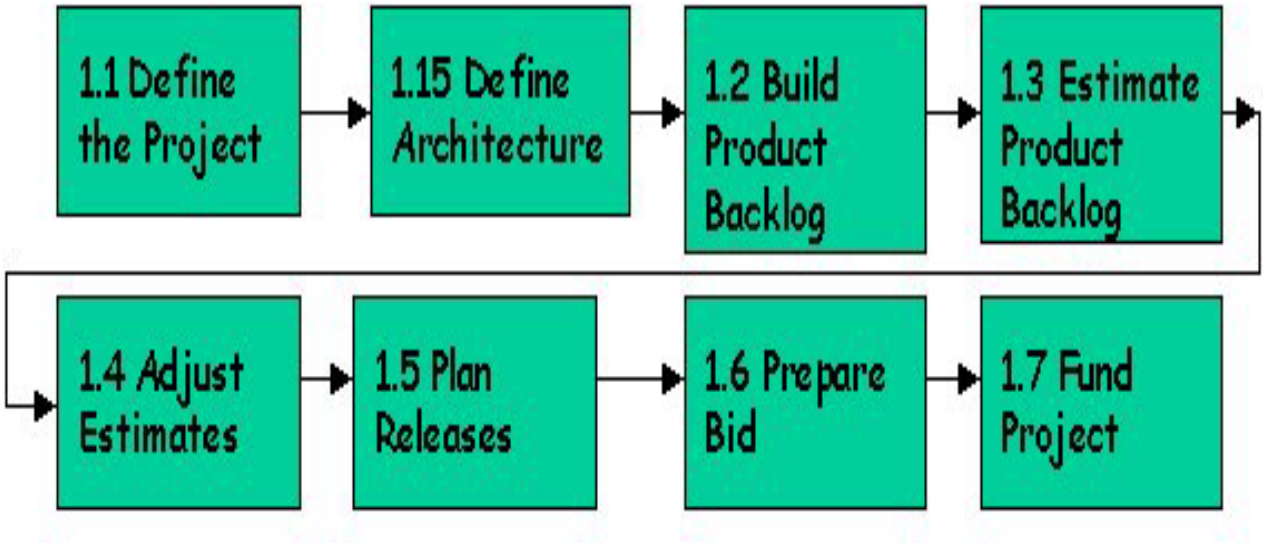
Planning Paths

Path	Type	Characteristics
1A	New, unfunded projects	Establishes baseline architectures, domain models, product backlog, requirements and vision. Produces enough detail to acquire funding.
1B	New, funded projects	Establishes initial product backlog to drive Sprints
1C	Already underway and funded projects	Establishes initial product backlog to drive first Sprint
1D	Fixed price/fixed date contractual development	

Path Name: *1A New, Unfunded Projects*

Newly conceived projects require funding. Before anyone will fund a project, they must understand the return on investment (ROI) of the project so they can evaluate this project against all other activities that are also competing for capital and funding. In order to assess the accuracy of the asserted ROI, they also need to understand enough details about the project as to determine its vision, risks, and underlying assumptions.

The purpose of the plan for a new, unfunded project is to lay out a investment vision against which management can assess and frequently adjust its investments, to lay out a common set of understandings from which emergence, adaptation and collaboration occur, and to establish expectations that reports will be measured against.



1A New Unfunded Project

Path Name: *1B New, Funded Projects*

Some projects have already been approved and funded, but have yet to get underway. Or, perhaps a project has tried to get underway but the complexity of the technology or requirements have precluded any progress.

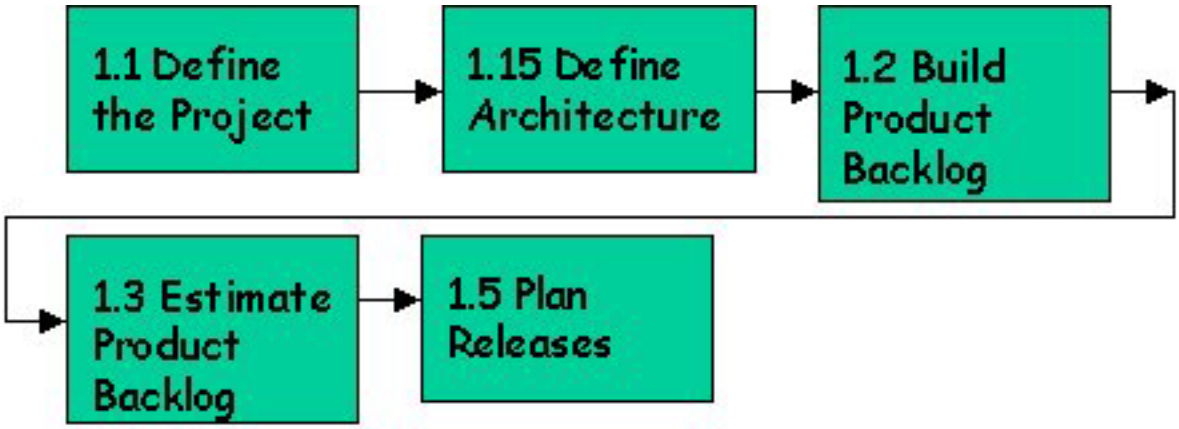
In this case, find someone who can represent the customers and users. Ask this person to take on the role of Product Owner and to identify some initial, high priority requirements. Then appoint a ScrumMaster and start by conducting Daily Scrums. A starter Product Backlog consists of some business functionality and the technology requirements. To implement this functionality, the team designs and builds an initial system framework with the selected technology. The team implements user functionality into this framework. The team may have to connect the functionality to a preliminary or existing database. Under these circumstances, the goal for the first Sprint is:

"*Demonstrate a key piece of user functionality on the selected technology.*"

When the team defines the Sprint Backlog to meet this goal, it includes tasks necessary to build the development environment, set up the team, define code management and build management practices, implement the target system technology on a test platform, and build the functionality. This constitutes a pretty full Sprint

This initial Sprint has two purposes. First, the team needs to settle into a development environment in which it can construct functionality. Second, the team builds a working part of the system to demonstrate to the Product Owner and customers within thirty days. Demonstrating functionality this quickly invigorates Product Owner and customer involvement and thinking. They realize the system is for real right now and think, "Now that the system is really being built, we'd better decide what do I want from it. I'd better get involved!" The first Sprint gets the team, Product Owner, and customers into a regular thirty-day rhythm of defining and delivering, defining and delivering.

While the team is working on the first Sprint, the Product Owners builds more Product Backlog. The Product Backlog doesn't have to be complete; it only needs to include enough top priority items to drive the next few Sprints. As the Product Owner and customers get a feel for Scrum, they start taking a longer view of Product Backlog. If a system or product vision isn't available, the Product Owner and customer will forge one. They will then construct Product Backlog on the basis of this vision.



1B New Funded Project

Path Name: *1C Underway, Already Funded Projects*

Sometimes you want to use Scrum to get an existing project or product development focused, productive, and generating code. A team is often struggling with changing requirements and difficult technology. The team may not have built any functionality yet, but instead has tried to deliver requirements documents or business models. A development environment already exists and the team is familiar with the targeted technology.

In this case, find someone who can represent the customers and users. Ask this person to take on the role of Product Owner and to identify some initial, high priority requirements. Then appoint a ScrumMaster and start by conducting Daily Scrums. Find out what is impeding the team. These initial Daily Scrum meetings may go on for hours. The team talks out its problems, including why it can't build software and often how frustrated it is. Challenge the team: "What can you build in the next thirty days?" You want to see team work together to build something, to prove that it can develop software. Try to get the team to focus on functionality that is important to the Product Owner. What really impresses the Product Owner, though, is that the team can build something at all within thirty days. In many instances, the team has gone for months without producing any functionality and the Product Owner and customers have given up. The most immediate goal is to get the team to believe in itself, and to get the Product Owner and customers to believe in the team. The Sprint Goal is:

"Demonstrate any piece of user functionality on the selected technology"

At the Daily Scrums, identify other impediments to the team's progress and help to remove them. If the team is able to build functionality during the first Sprint, the Product Owner and team collaboratively determine what to do next at the Sprint Review and Sprint Planning meeting. I have never had a team fail to meet this challenge.

1C Already Underway Funded Projects

Path Name: *1D Fixed Price/Fixed Date Projects*

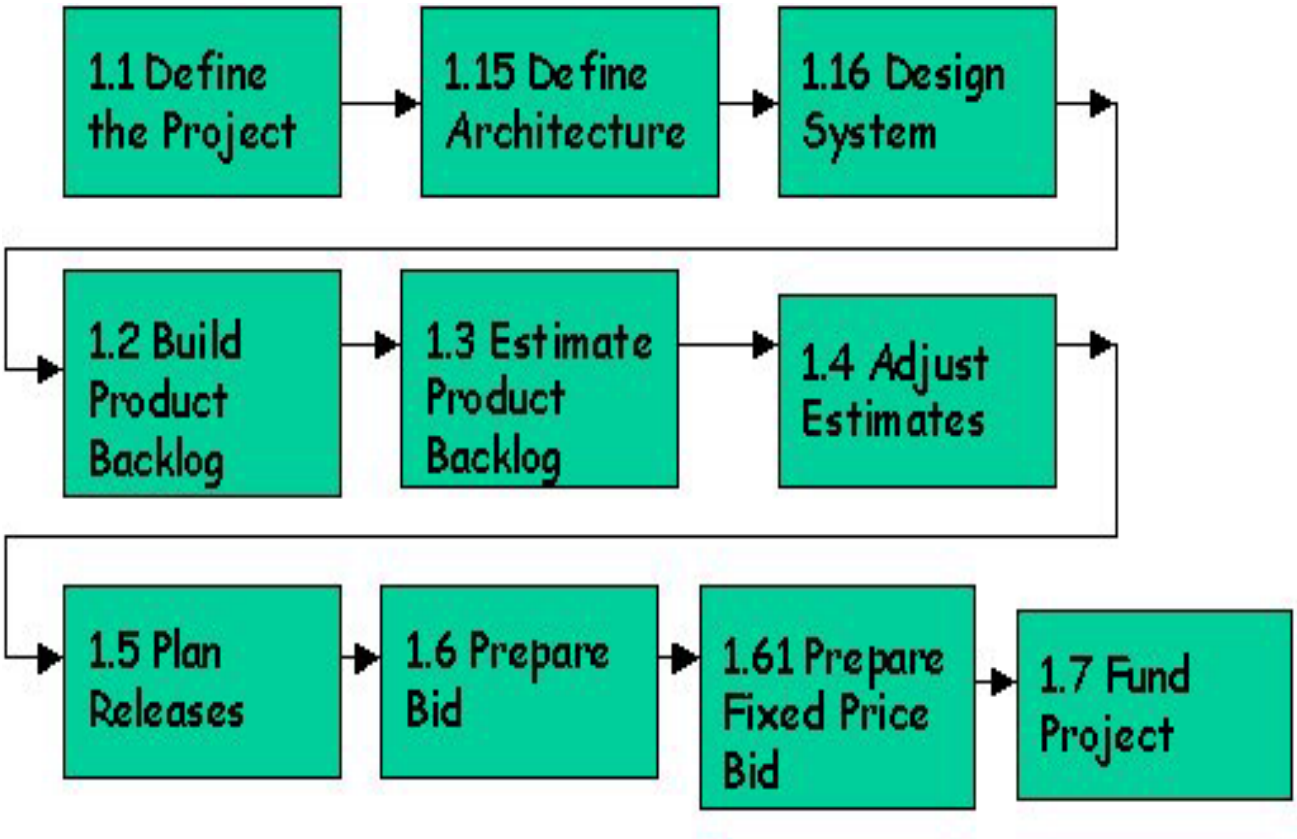
Fixed Price/Fixed Date projects are the consequence of the belief that the functionality, quality, delivery date, and cost for developing a complex system to meet complex requirements can be predetermined and contracted for between a customer and provider. If the technology proves more difficult than the provider expected, the provider is expected to eat the overrun. If the requirements need to change because the customer has to change their mind, the customer is expected to pay for this through "change control" mechanisms and additional funding. Both sides attempt to ensure that they will win, or if they lose that the loss won't be out-of-control.

Agile projects are constructed based on the acceptance that it is impossible to predict complex projects. The project starts and the customer and provider learn together. The goal of the project is not to deliver functionality, but to deliver business value. Agile projects openly accept the fact that much of the initially scoped functionality may be irrelevant to delivering business value and may be unused by the customer despite its cost. The unneeded functionality will also always require maintenance, sustaining its unnecessary costs well into the future.

However, many customers still think in terms of the fixed price/fixed date contract and they construct their request for proposals, evaluation of capabilities, and contracts in these terms. In these cases, an organization that is skilled in agile development needs to respond within the customer context and understanding, while providing room for the customer to learn and take advantage of agile principles **if they are so inclined**. Regardless, the practices in agile development will make for a more productive project with more functionality for the cost

The primary difference between planning a fixed price/fixed date contract and a new, unfunded project is the degree to which the system must be analyzed and specified prior to beginning the project. It is incumbent on any bidder, regardless of the development process they will employ, to fully understand the technical and business architecture for delivering the proposed system, and the detailed entities that will inhabit these architectures (classes, methods, sequences, interfaces, interactions).

Subject: 1D Fixed Price/Fixed Date Projects



1D Fixed Price/Date Contract

Activity ID: *1.1 Define the Project*

Responsible: *Product Owner*

Purpose: To describe the anticipated system in enough detail that it can be understood, communicated and justified for funding. Since the system will be developed empirically, using iterations to build the most appropriate increments of functionality, the project description lays out the vision of the product or system, not the detailed requirements. Goals are established for initial Sprints and releases, and functional and nonfunctional requirements that will attain those goals are enumerated, but not detailed.

Assumptions: The project has an advocate who is able to clearly articulate the purpose and vision of the system, as well as move this project plan through the funding process. This person often becomes the Product Owner.

Risks: Don't overspecify the system to be developed. The exact requirements that will satisfy the business needs will evolve as the Product Owner, stake holders, and development team learn more about the business needs, ways of satisfying them, the changing business environment, and the capabilities of the technologies.

Description:

The first step is to prepare the project definition. This is the vision that will be shared by the funders of the project and those developing the system. The project definition is a rallying point that provides context and focus to the project teams. This vision guides them as they make decisions about the functionality and design of the product. Scrum relies on emergence of details, so the project definition is kept to the conceptual and contextual. It should not be lengthy and definitive, but suggestive and guiding. It should set forth expected outcomes and possible risks and issues.

One suggested format for project definition is:

- **Vision.** What are we trying to do? What will the business operation or product look like when the project is completed? How will it be better and more valuable? Why is this worth doing? Why is this vision unique and valuable?
- **Business Operations.** How does this project change our business? How does this improve our business? How does this affect our competitiveness? How does this affect our bottom line?
- **Releases**. Identify the implementation plan for this vision. What software releases are required to support the implementation plan and what is their general capability and functionality? What changes to the business must be affected? What change management process will be employed to implement these changes.
- **Functionality and business architecture.** What is an overall business flow with this project implemented? Where are the significant changes? What functionality is required to make these changes? To what degree do the significant stakeholders and others impacted by these changes agree to the vision?

Activity ID: *1.1 Define the Project*

- **Target Technical architecture.** What is the architecture of the technology that will implement this required functionality and business architecture? What is the reliability and stability of this technology, both individually and operating as a system?
- **Development infrastructure and technology.** What is the development platform that will be used to develop this system? What is the reliability and stability of this development platform?
- **Return on Investment.** What benefits will be accrued by the business as a result of this project? What measures will be performed to assess these benefits as the project functionality is implemented? How will these measures be isolated from affects other than this project? What are the costs of this project? How will they be measured?
- **Development.** How will these products be developed? Who will do the development, how will the teams be arranged, where will the engineers be recruited or located, how will quality be assured, how will maintainability, sustainability and enhancability be addressed, and what development process will be followed? List all assumptions and risks.
- **1st Year Plan.** How do you plan to execute for the first year? Identify milestones, work backlog, and product releases. Prepare spreadsheets of all expenses and revenues. Identify each line item. Summarize for every quarter. Identify all assumptions and risks by quarter.
- **2nd Year Plan.** How do you plan to execute for the second year? Identify milestones and product releases. Prepare spreadsheets of all expenses and revenues. Identify each line item. Summarize for every half-year. Identify all assumptions and risks by half-year.

Questions that need to be answered in the project definition include: Will the project improve productivity? Will the project help increase user satisfaction and increase the number of return buyers? Will this project increase the number of repeat sales? Will this project improve market share? Will this project help reduce the prices from suppliers? Why should this project be undertaken? What is this project all about? How will this project create the results that are expected and how will those results be measured? How much will the project cost? What is the schedule for costs and returns on investment?

Activity ID: *1.15 Define Architecture*

Responsible: *Product Owner, Team*

Purpose: Architectures describe the business aspects and technical aspects of the planned system. They are prepared using modeling or drawing tools at the organization and represent a vision of the planned system. The business architecture is used for implementation planning. The systems architecture is used for coordinating development if multiple teams are used. Both are prepared with enough detail to guide decision making and management of the project. The architecture diagrams are also used in project reporting to demonstrate the parts of the system developed each Sprint.

Assumptions: Business and technical staff with adequate domain knowledge are available to the Product Owner, to collaborate in creating architectures that will successfully deliver the system visions.

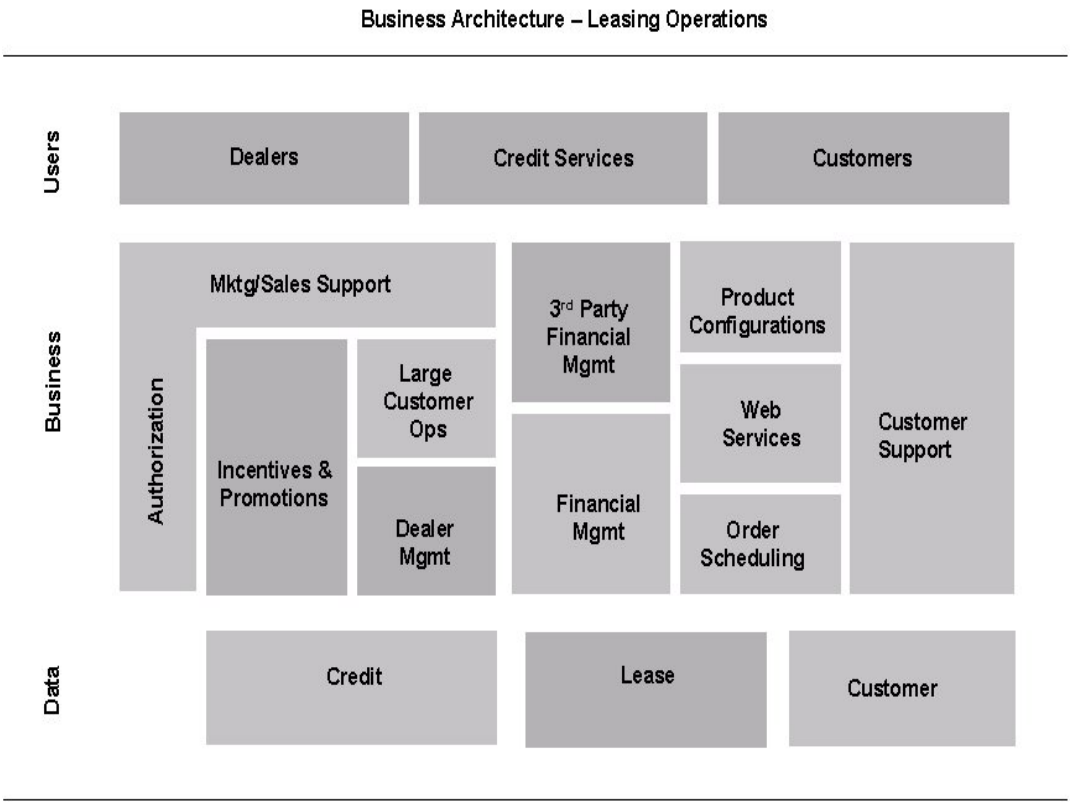
Risks: The architectures are developed in too much detail. If this is done, excessive work is being done in the planning phase. Also, this architecture detail will stifle and restrict emergence as the project progresses. Detailing these architectures should be done during the project Sprints, as the system emerges.

Description: To describe in business and technical terms the overall components, interactions, and flows of the system. The models used to describe the architectures are for informational, decision making, and informational purposes. The models are not used to decompose the system progressively into detailed code. That type of decomposition is not used in the Scrum development process.

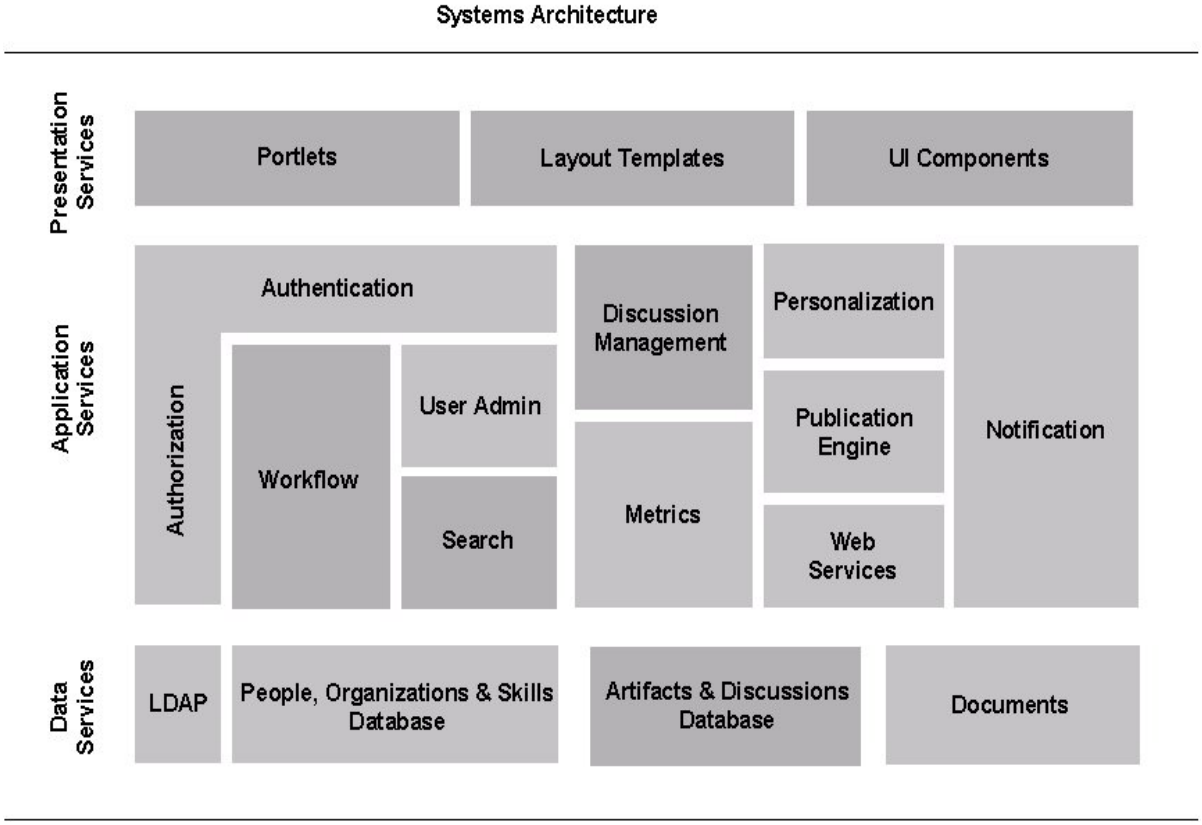
Both the systems and business architectural models are used to envision the system as it is intended to be operated and implemented. Both of the models can also be used to track the progress of the project by highlighting progressively the parts and components that are completed or partially completed every Sprint. This augments the Product Backlog and ROI reporting at a visual level.

Activity ID: *1.15 Define Architecture*

An example model of a business architecture is:



An example model of a systems architecture is:



Activity ID: ***1.16 Design System***

Responsible: ***Product Owner, Team***

Purpose: To further detail the architectures of the system so that detailed costs estimates of the system can be constructed. The designs are complete both to entities, classes, methods, sequences, interactions, and subsystem development. Enough detail is provided to understand what the system is and how it will be constructed.

Assumptions: Funding cannot be acquired without an initial estimate of costs and that this estimate of costs will rely on change control or detailed collaboration mechanisms to balance emergence with cost and date targets.

Risks: The cost of this effort exceeds that which would occur during iterative, incremental delivery of design since this cost is sunk regardless of changes. This type of development only can avail itself of the benefits of increments and iterations, while the benefits of collaboration and emergence are minimized.

Description: Decompose the architectural models previously constructed into all of the detail artifacts required to create an effective estimate of system development.

Activity ID: *1.2 Build Product Backlog*

Responsible: ***Product Owner***

Purpose: To develop an initial list of the what the product will consist of. This includes functionality, such as "check balance prior to disbursing cash", to non-functionality, such as "system will provide subsecond response time when 100,000 simultaneous users access functionality", to environmental, such as "multiple distributed teams will need a shared team environment."

Assumptions: The product backlog items can be derived from the vision of the system or product. The items that are most important should be prioritized to the top of the list, with items of diminishing importance ranked accordingly. Think through the higher priority product backlog in more detail, breaking down items until they are well understood.

Risks: The product backlog should be painted in broad strokes. Only detail the items that are of highest priority, that require delivery first to demonstrate the value and probability that the product or system can actually be built. Lower priority functionality should be coarse grain; high priority functionality should be fine grain.

Description:

The product backlog is a list of functions, features, technology, and capabilities that the product might have. This is not a task list, but a list of things that require work to be implemented in the product or system. Everything that can be thought of for the product is in this list. The list is emergent, with some items disappearing and others popping up over time. Emergence happens because more is known about the product as it is developed, and the business environment within which the product will be implemented changes.

The product backlog is prioritized. Higher priority items usually have a higher ROI potential than lower priority items. Sometimes lower priority product backlog receives a higher priority if it must be implemented first, indicating a dependency of an otherwise higher priority item. Product backlog items often change priority over time to reflect the impact of emergence. As the users see product functionality, they often change their minds about how the functionality works or what they want next.

No item is too insignificant to be on the product backlog; however, some items are very low priority compared to others and may never be implemented.

The product backlog consists of three types of items:

1. Product functionality - what functions could the system perform to deliver the value anticipated in the product vision.

2. Nonfunctional requirements - for the product to deliver the necessary value, what operational aspects must it demonstrate, such as performance, security, reliability, and cost requirements.

3. Environmental requirements - what capabilities and environments must be in place for the product to be developed and delivered. For instance, if the teams have never developed n-tier object-oriented systems previously, significant training, tool investment, and development environment investment must occur early in the development iterations.

Inspect the product vision. Bring together all of the stakeholders of the system and brainstorm what is expected of the product. Write down all of the items. When more details are available about an item, reference those details so the they can easily be available to the team. Keep at this work until you are comfortable that a pretty good description of the requirements for the system are listed.

This list is skeletal. This activity is not intended to analyze the system, merely to scope and list its requirements. The actual analysis is performed during the iterations of work performed later in the project. This activity queues up "what" the system should look like. The development teams will analyze exactly how this "what" operates.

While prioritizing the product backlog, ensure that nonfunctional requirements are high priority. This ensures that the early functionality is developed within the context of the operational and nonfunctional requirements of the system.

While prioritizing the product backlog, ensure the environmental requirements are high priority. The team requires an environment within which to build, test, and deploy the system.

Figure 4 is an example of some product backlog developed to start the project to implement the website for the AgileAlliance organization.

Fig 4

Priority (1-9)	Function	Full Description	Raw Dev. Effort
1	General	Setup development environment	4
1	General	Confirm use of zope as development environment	2
1	Membership	Ability to sign up for various level of membership	4
1	Membership	Ability to use credit card to pay for membership	15
1	Membership	Provide extract from database to external sources	3
1	Membership	Notify members with membership data	5
1	Membership	Generate receipts and certificates	5
1	Membership	Tie membership program to bank accounts	5
1	Membership	Implement open access database (MySQL?)	6
2	News	Authoring environment for news	8
1	Website	Website look, feel, navigation, initial pages	10
3	Founders Page	Include agilealliance.org founders page	10
3	Sponsors	Display sponsors and link to web sites	6
4	Articles	Authoring environment for articles	10
4	Articles	Organizing and sorting capability for articles	4
4	Articles	Library catalog for articles	8

Activity ID: *1.3 Estimate Product Backlog*

Responsible: *Product Owner, Team*

Purpose: To gain an understanding of the amount of effort to develop the product or system. As estimates are thought through, a more detailed understanding of each product backlog item is required. Make more effort to reach this estimate for higher priority than lower priority items. The lower priority items may change prior to being developed.

Assumptions: People with similar skills to the team that will be developing the product or the team itself are available to assist the product owner. The estimate emerges in a dialog between them, as the product owner describes the requirements in as much detail as is required to make the estimate. These details are captured and related to each backlog item.

Risks: Too much time is spent estimating. The goal of the estimates is to gain a general understanding of the cost of the system or product. This is used to determine whether it is economic to develop it. Do enough, but not too much.

Description:

The Product Owner work with the business departments and the development organization to develop estimates for the amount of time it will take to build the functionality. These estimates include the effort to analyze, design, develop, document, and test the functionality. The estimates are usually in person days.

Use the people who will be staffing the project teams to make the estimates. If they aren't known yet, have people of equal skills and project domain knowledge make the estimates. Do not have experts or outsiders make the estimates. Their estimates are irrelevant to those who will actually be doing the work.

The accuracy of the estimates decreases as the priority the requirement in the product backlog decreases. Higher priority product backlog is more important to the business, will be developed sooner, is likely to have been thought through, and is likely to be implemented. Lower priority product backlog items are more susceptible to change. Before they rise to the top of the product backlog for selection in a Sprint, business conditions are likely to change, the users are likely to change their minds, and the technology may shift. Less time is spent trying to think through the lower priority product backlog because it is less likely to happen. Scrum project management tries to focus efforts on work at hand, only keeping a general view of future work for context.

Activity ID: *1.3 Estimate Product Backlog*

Estimating is iterative. Estimates change as more information emerges. If the project managers can't get a believable estimate for a top priority backlog item, its priority should be lowered. Since not enough is known about the functionality to create a believable estimate, usually not enough is known about it to start work. As an alternative, the item can be kept as a high priority, but reclassified as an issue. An example of an issue is, "Salesmen need to respond to orders with estimated ship dates within ten minutes; what functionality is needed to support this requirement?" The work to turn this issue into required functionality might take ten days; estimate the issue at ten days. If an estimate for functionality predicted for the next six months has an estimate greater than 10 days, break it down into multiple product backlog items

Lower priority product backlog is vague, a placeholder so that the team thinks more about it as its priority increases. The estimates are not very reliable, only reflecting the level of thought the team puts into understanding how that functionality might be implemented on the initially selected technology. Estimates often range from ten to forty days for lower priority backlog items.

Figure 4 is an example of some product backlog developed to start the project to implement the website for the AgileAlliance organization.

Fig 4

Priority (1-9)	Function	Full Description	Raw Dev. Effort
1	General	Setup development environment	4
1	General	Confirm use of zope as development environment	2
1	Membership	Ability to sign up for various level of membership	4
1	Membership	Ability to use credit card to pay for membership	15
1	Membership	Provide extract from database to external sources	3
1	Membership	Notify members with membership data	5
1	Membership	Generate receipts and certificates	5
1	Membership	Tie membership program to bank accounts	5
1	Membership	Implement open access database (MySQL?)	6
2	News	Authoring environment for news	8
1	Website	Website look, feel, navigation, initial pages	10
3	Founders Page	Include agilealliance.org founders page	10
3	Sponsors	Display sponsors and link to web sites	6
4	Articles	Authoring environment for articles	10
4	Articles	Organizing and sorting capability for articles	4
4	Articles	Library catalog for articles	8

Activity ID: *1.4 Adjust Backlog Estimates*

Responsible: *Product Owner*

Purpose: To adjust the product backlog estimates to reflect any factors reduce team effectiveness. Scrum assumes an optimal work environment. This activity forces an understanding that suboptimal product factors have a cost.

Assumptions: This activity is not mechanical, plugging in factors. Instead, it is intended to provide cost awareness. Every decision regarding the team working environment has an impact on their productivity and an attendant cost. If the management is aware, this activity is merely a reminder of these costs. If management is not aware of these costs, this activity provides a monetary cost to suboptimal environments.

Risks: Don't quibble about the factor values. They are generalized and not intended to be precise for every situation.

Description: The estimated time for each product backlog item can be adjusted by a "complexity factor." The complexity factor reflects the degree of complexity that will affect the estimates. In one instance, the product owner doubled all estimates because the team was completely new and had never even worked at the company prior to this project.
The guidelines for adjusting estimates to reflect complexity are just that: guidelines. This is not a precise science. Complexity adjustments are used solely and simply to get the person estimating the work to consider the complexity.

Two factors that complicate the product backlog are requirements and technology. How well are the requirements for that product backlog item known? How much agreement is there about what the design and implementation of that item? How difficult and stable is the implementation technology? Some product owners believe that they take these factors into account when making the initial item estimate. It is preferable to keep this part of the estimation separate. The appropriate complexity factor for each product backlog item can be selected from Figure 5 and put it into the complexity

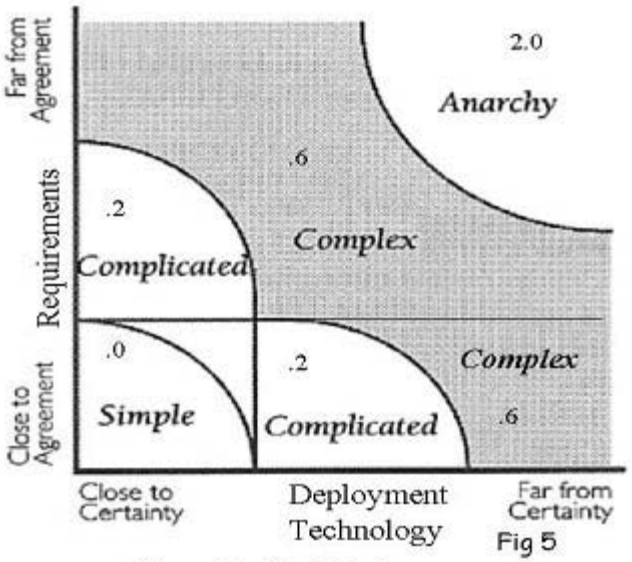


Fig 5
Complexity Factor

column for each item. Or, a general complexity factor can be determined and applied to all product backlog items.

Generally speaking, the complexity factor should only be applied to a known product backlog horizon. That is, if it is possible that the team environment will change, don't apply complexity factors past that horizon. Diminish the impact of the complexity factors as the team can be expected to master the technical and business domain.

Three more complexity factors are added to items in the complexity factor column. These reflect the drag on team velocity, the adequacy of the working environment, and overhead for multiple team projects. Again, these are guidelines for adjusting estimates to reflect complexity. This also is not a precise science. Complexity adjustments are used solely and simply to get the person estimating the work to consider what is negatively affecting team productivity and to prompt the project managers to consider making changes. The appropriate drag these complexity factors put on productivity can be selected from the table in Figure 6.

Drag	# of years together	Knowledge of technology	Knowledge of domain
0.8	< 3 months	Low	Low
0.75	< 3 months	Low	Medium
0.7	< 3 months	Low	High
0.75	< 3 months	Medium	Low
0.5	< 3 months	Medium	Medium
0.5	< 3 months	Medium	High
0.75	< 3 months	High	Low
0.5	< 3 months	High	Medium
0.35	< 3 months	High	High
0.6	< 1 year	Low	Low
0.55	< 1 year	Low	Medium
0.5	< 1 year	Low	High
0.55	< 1 year	Medium	Low
0.3	< 1 year	Medium	Medium
0.25	< 1 year	Medium	High
0.5	< 1 year	High	Low
0.25	< 1 year	High	Medium
0.2	< 1 year	High	High
0.5	> 1 year	Low	Low
0.45	> 1 year	Low	Medium
0.4	> 1 year	Low	High
0.45	> 1 year	Medium	Low
0.35	> 1 year	Medium	Medium
0.2	> 1 year	Medium	High
0.4	> 1 year	High	Low
0.2	> 1 year	High	Medium
0	> 1 year	High	High

Fig 6

Drag on team productivity: The project managers estimate that the team will take 7 days to turn a product backlog item into working functionality. However, this productivity is decreased when teams haven't worked together before, when they are not familiar with the technology that they will be using, and when they aren't familiar with the business domain. Figure 6 provides an estimate of the amount of drag these factors can have on team productivity.

Activity ID: *1.4 Adjust Backlog Estimates*

Adequacy of working environment: Scrum works best when everyone on a team is together in an open environment with adequate work and conference rooms. Productivity is adversely affected when this type of an environment isn't available. Although the effect is variable (separate doors with offices in different parts of the building and city are worst!), a 0.2 factor to the complexity column is usually appropriate to add if the environment is sub optimal.

Multiple teams: The last adjustment is based on management and communications overhead caused by multiple teams working together on the same project. Despite best efforts to group a team's work to maximize cohesion and minimize coupling to other teams, an additional factor of 0.1 can be consider when there is more than one team.

Summing The Impact: All of these complexity factors are summed, added to a 1.0 base, and multiplied by the raw effort. The overall estimate of work for a release or the entire product backlog is the sum of these multiple product backlog items.

Ó(raw effort * (1.0 + complexity factor + drag + working environment + multiple teams)

The product backlog spreadsheet for the AgileAlliance website in Figure 7 is updated for complexity.

Fig 7

Priority (1-9)	Function	Full Description	Raw Dev. Effort	Cplxty	Est Dev. Effort	Sprint #	Status
1	General	Setup development environment	4	0.0	4		
1	General	Confirm use of zope as development environment	2	0.0	2		
1	Membership	Ability to sign up for various level of membership	4	0.2	4.8		
1	Membership	Ability to use credit card to pay for membership	15	0.2	18		
1	Membership	Provide extract from database to external sources	3	0.2	3.6		
1	Membership	Notify members with membership data	5	0.2	6		
1	Membership	Generate receipts and certificates	5	0.2	6		
1	Membership	Tie membership program to bank accounts	5	0.0	5		
1	Membership	Implement open access database (MySQL?)	6	0.0	6		
2	News	Authoring environment for news	8	0.0	8		
1	Website	Website look, feel, navigation, intial pages	10	0.4	14		
3	Founders Page	Include agilealliance.org founders page	10	0.2	12		
3	Sponsors	Display sponsors and link to web sites	6	0.0	6		
4	Articles	Authoring environment for articles	10	0.2	12		
4	Articles	Organizing and sorting capability for articles	4	0.2	4.8		
4	Articles	Library catalog for articles	8	0.6	12.8		

Activity ID: *1.5 Plan the Releases*

Responsible: *Product Owner*

Purpose: To realize benefits from a system after developing some of its intended functionality, releases are planned to implement levels of functionality that will provide users and customers with incremental benefits that outweigh the cost of implementation.

Assumptions: The system consists of functionality that can be parsed into releases that can be logically absorbed by the customers and users.

Risks: Packaging functionality into a release costs money and effort, both to the development teams and to the customers and users that have to implement and learn how to use it. Ensure that the benefits from releases more than offset the costs.

Description:

The Product Owner identifies the functionality that will probably constitute system releases. The functionality is a meaningful set of related functionality that will cause a business change or provide a useful set of customer functionality. The business change is expected to improve operations, to reduce costs, or to improve revenues. The release implementation date assumes specific business conditions for these improvements or benefits to occur.

To project these releases, The Product Owner groups the product backlog into several releases. The date of each release is then estimated by dividing the effort cost per day into the total estimated days of effort to develop and implement that release's functionality.

(Adjusted effort to develop and implement selected functionality)/ (Available effort (number of developers * productive hours/day)

The Product Owner then adjusts the functionality or available effort to create a release with functionality on a date when the return on investment can be expected to create the expected value.

Activity ID: *1.5 Plan the Releases*

A visual depiction of the release plan superimposed on the product backlog is shown in Figure 8. This is done by inserting a line that says “release” after the functionality. All functionality between that line and the prior release line (or top of the product backlog) is planned for that release.

The release plan is only an estimate. At the end of every actual Sprint, these lines will be adjusted based on actual team productivity and any changes in business conditions.

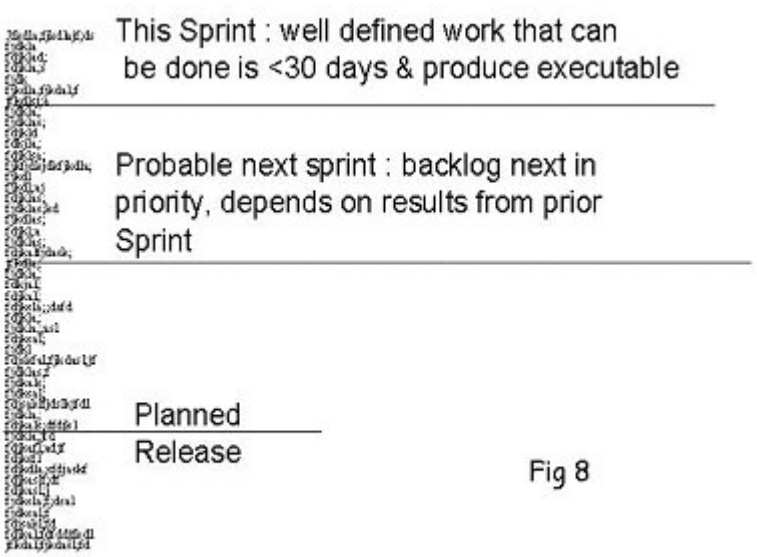


Fig 8

Activity ID: *1.6 Prepare Bid*

Responsible: *Product Owner*

Purpose: The bid consists of spreadsheets that lays out the costs and the benefits. The assumptions that have been instrumental in laying out the plan, vision, architectures, and estimates are now made explicit. The details underlying the expectation of realization of benefits are also made explicit. In the detailed analysis of the viability of proceeding with the project, the spreadsheet of costs and benefits with all underlying assumptions quantified is essential.

Assumptions: The product owner has performed adequate analysis of the marketplace, customer operations, and users to accurately quantify benefits. The Product Owner is adequately aware of the costs of development, including level and cost of personnel and overall ongoing costs of technology, to quantify assumptions. That the Product Owner is able to quantify ongoing maintenance, sustenance, and enhancement costs which will inevitably offset long term benefits.

Risks: The bid is as good as the competence of the Product Owner and the domain experts available to him or her in preparing the bid.

Description: The bids are somewhat different if the system will be a commercial product developed by an Independent Software Vendor (ISV), or an internal development organization for the consumption of the internal organization. This activity discusses the spreadsheets required for an ISV. With some minor adjustment of terms, the same spreadsheets are appropriate for internally consumed systems.
Prepare the following spreadsheets:

- Spreadsheet 1: revenue and expense projections for the development, deployment, enjoyment, and operation of the system throughout the customer base. For an internal system, benefits and costs replace revenues and expenses.

- Spreadsheet 2: a list of assumptions and quantification of the various assumptions underlying the costs and benefits.

- Spreadsheeet 3: an analysis of the costs and benefits in alternate scenarios. Each scenario identifies the quantity for one or more specific assumptions and shows how the development and implementation of the system will play out under those scenarios.

The spreadsheets on the following pages represent examples from the book, “Lean Software Development: An Agile Toolkit”, Mary Poppendieck Tom Poppendieck, to be published by Prentice Hall in June, 2003. These fine examples are in the Scrum methodology through the permission of the Poppendiecks.

Activity ID: *1.6 Prepare Bid*

Spreadsheet 1: cost benefit analysis for development and operation, or expense revenue analysis for development and sale.

	Assumptions	Year -2	Year -1	Year 0	Year 1	Year 2	Year 3	Year 4	Year 5
Revenue									
Average Selling Price	Decreases 10% / yr			$ 1,000	$ 900	$ 810	$ 729	$ 656	$ 590
Total Market Units				10,000	20,000	40,000	60,000	40,000	20,000
Market Share				30%	40%	50%	50%	50%	50%
Units Sold				3000	8000	20000	30000	20000	10000
Total Revenue				$ 3,000,000	$ 7,200,000	$ 16,200,000	$ 21,870,000	$ 13,122,000	$ 5,904,900
Expense									
Unit Mfg & Distribution Cost	Decreases 5% / yr			$ 200	$ 190	$ 181	$ 171	$ 163	$ 155
Unit Warranty & Support Cost	Decreases 10% / yr			$ 200	$ 180	$ 162	$ 146	$ 131	$ 118
Total Unit Cost				$ 400	$ 370	$ 343	$ 317	$ 294	$ 273
Manufacturing/Support Cost				$ 1,200,000	$ 2,960,000	$ 6,850,000	$ 9,518,250	$ 5,882,425	$ 2,728,542
Gross Margin $				$ 1,800,000	$ 4,240,000	$ 9,350,000	$ 12,351,750	$ 7,239,575	$ 3,176,358
Gross Margin %				60%	59%	58%	56%	55%	54%
Development		$ 2,000,000	$ 2,000,000	$ 1,500,000	$ 750,000	$ 750,000	$ 500,000	$ 500,000	$ 500,000
Marketing	15% of sales			$ 450,000	$ 1,080,000	$ 2,430,000	$ 3,280,500	$ 1,968,300	$ 885,735
G&A	5% of sales			$ 150,000	$ 360,000	$ 810,000	$ 1,093,500	$ 656,100	$ 295,245
Total Expense		$ 2,000,000	$ 2,000,000	$ 3,300,000	$ 5,150,000	$ 10,840,000	$ 14,392,250	$ 9,006,825	$ 4,409,522
Profit (Loss)				$ (300,000)	$ 2,050,000	$ 5,360,000	$ 7,477,750	$ 4,115,175	$ 1,495,378
% of Revenue				-10%	28%	33%	34%	31%	25%
Cumulative Revenue				$ 3,000,000	$ 10,200,000	$ 26,400,000	$ 48,270,000	$ 61,392,000	$ 67,296,900
Cumulative Expense		$ 2,000,000	$ 4,000,000	$ 7,300,000	$ 12,450,000	$ 23,290,000	$ 37,682,250	$ 46,689,075	$ 51,098,597
Cumulative Profit		$ (2,000,000)	$ (4,000,000)	$ (4,300,000)	$ (2,250,000)	$ 3,110,000	$ 10,587,750	$ 14,702,925	$ 16,198,303
Cumulative Profit % of Revenue				-143%	-22%	12%	22%	24%	24%

Activity ID: ***1.6 Prepare Bid***

Spreadsheet 2: itemization and quantification of assumptions underlying cost benefit spreadsheet.

		Assumptions	
# calls per day	10,000		
av. minutes per call	0.53		
total time (hours) per day	88		
peak call rate	8.2		
staff utilization	75%		
required staffing level	14		
average hourly pay	$7.50		
regular hours in month	176		
total regular monthly pay	$18,480		
% overtime	19%		
$ overtime	$1,980		
total base pay	$20,460		
supervision	$2,455	12%	base pay
benefits	$7,161	35%	base pay
turnover	2		
training	$840	7	days
total salary and benefits	$30,916		
system admin	$10,000		
content maintenance	$5,000		
hardware	$6,500		
facilities	$2,960		
total monthly cost	$55,376		
revenue	$60,000		
profit	$4,624		
margin	8%		
customer satisfaction	92%		
first call resolution	58%		
abandoned calls	3.60%		
downtime hours	3.5		

Activity ID: ***1.6 Prepare Bid***

Spreadsheet 3: Summary of cost benefit model under varying conditions, with the conditions identified by assumption

	Baseline	Goal 1: New Operating Sustem	Goal 2: $10,000 in new business	Goal 3: 10% lower time per call	Goal 4: 50% less training required	One time savings for 50% reduction in training
Revenue	$ 60,000	$ 60,000	$ 70,000	$ 60,000	$ 60,000	
Cost						
Call Center Staffing	$ 30,916	$ 30,916	$ 36,737	$ 26,615	$ 30,496	
Support Staffing	$ 15,000	$10,000	$ 15,000	$ 15,000	$ 15,000	
Hardware & Facilities	$ 9,460	$ 8,000	$ 9,460	$ 9,460	$ 9,460	
Total	$ 55,376	$ 48,916	$ 61,197	$ 51,495	$ 54,956	
Profit	$ 4,624	$ 11,084	$ 8,803	$ 8,505	$ 5,044	
Profit Margin	7.7%	18.5%	12.6%	14.2%	8.4%	
Monthly Benefit		$ 6,460	$ 4,179	$ 3,881	$ 420	
One Time Benefit						$ 2,940

Activity ID: *1.61 Prepare Fixed Price/Date Bid*

Responsible: *Product Owner*

Purpose: Bids are sometimes required to include a fixed date of delivery and fixed costs for delivery. Such bids usually require a detailed list of functionality and the architecture for delivering the functionality, along with all underlying assumptions. The bid spreadsheets developed in Activity 1.6 are the basis of such bids. However, for a fixed price/fixed date bid they are much more extensive. All of the costs are specified along with all of the functionality. The artifacts underlying the functionality must also be assessed, measured, and quantified.

Assumptions: The mechanisms for preparing fixed price/fixed date bids traditionally is applicable to agile development processes and Scrum. However, rather than bidding at the task level by including costed out PERT charts, the Scrum bid assesses workload at the requirements and nonfunctional requirements level.

Risks: Although Scrum is an adequate mechanism for addressing fixed price/fixed bid contracts, the Scrum methodology doesn't contain any processes or mechanisms for change control. Change control is the mechanism for modifying the emerging expectations (sometimes called Scope Creep) of the customer. This is particularly important with Scrum because of the incremental delivery and frequent interaction with the customer with live functionality.

Description: To prepare a fixed price/fixed date bid that, perform the following:

- Develop vision, value statement with prospect.
- Create product backlog of functional requirements and nonfunctional requirements to meet system and application characteristics.
- Insert release backlog.
- Prioritize product backlog
- Review and revise with customer in light of vision and value statements.

Then,

- Create enough architecture and design to develop estimates for each product backlog estimate.
- Use all estimating and scaling factors described in "Planning the Release."
- Discuss with prospect how value will be delivered incrementally.
- Discuss with prospect that they are permitted to change product backlog content and priority in collaboration with the team as long as total estimates stay the same.
- Submit bid based on product backlog.

Through the above process, the prospect is educated to the opportunities of agile development, where they can iteratively select top priority functionality development to incrementally deliver value. The prospect that a subset of the identified functionality can deliver the majority of the value to the prospect is another possibility that should be presented. If this is successful, the bidder and the prospect can enter into a collaborative, win/win relationship rather than the contractual win/lose relationship of a fixed price/fixed date contract.

Activity ID: *1.7 Fund Project*

Responsible: *Product Owner*

Purpose: The plan is presented to the funding source, such as an organization capital committee or a venture capitalist. The plan has enough detail for the funder to understand the system and its intended purpose, as well as derive enough confidence in the plan and the Product Owner to justify the funding of such a project. As part of this activity, the Product Owner constructs ROI gauges that will be used to report on the incremental delivery of benefits as compared to costs so that the funding source can track the planned return on investment.

Assumptions: The funding source believes that technology projects are manageable and that plans can be executed through he use of iterative, incremental techniques in the Scrum agile process. Some education may have to be provided about such processes are part of presenting the plan.

Risks: Particularly in such hierarchical environments as Department of Defense procurement, the agile concepts that lend themselves to ROI measurements and emergent planning are unknown. These environments may be inappropriate for establishing ROI gauges.

Description:

As the project progresses, reports on its successful delivery of functionality are expected by the funding source. Since funding is usually provided in expectation of offsetting benefits and adequate return on investments, a way of measuring, tracking, and reporting these benefits and costs is developed, used to gain funding for the project, and used to measure the success of the development and successive implementations of releases of the system.

Phase Name: *2 Define Increment of Shippable Product*

Every Sprint creates an increment of functionality that is potentially shippable. The contents and characteristics of this increment depends on the type of system and functionality that is being developed. This activity defines the standards, conventions, templates, and contents of each product increment.

For more details, see the description in the Artifact section of the methodology.

What is a Potentially Shippable Product Increment

A number of people have questioned how Scrum and other agile processes support mission critical, FDA approved, or other varieties of software systems. These questions have arisen because the vanilla version of the most popular agile processes describe the construction of software that is single use; that is, it is used within the organization that develops it, with the development mostly done by the internal Information Technology (IT) organization.

Most agile processes require development teams to built an increment of product functionality every iteration, or Sprint. Scrum and Extreme Programming require that this increment be potentially shippable. This usually requires that the increment consist of thoroughly tested code that has been built into an executable, and the user operation of the functionality is documented, either in Help files or user documentation.

If the product increment that is created during the Sprint has more exacting use, the development organization usually defines the additional product requirements as standards or conventions. For example, the Federal Drug Administration (FDA) must approve a product that will be used in life-critical circumstances by in numerous health care settings if the product is to be used in the United States. As part of the approval process, the FDA checks that specific information regarding the product is provided, such as requirements traceability and specific functionality operation. For each increment to be potentially shippable, these additional facets of the product must also be developed – so that each increment of the product is potentially ready for FDA approval.

Similarly, some products require that performance requirements be modeled and the performance demonstrated mathematically, not just through statistical measurement of the actual system. The model with all required rigor is thus an additional part of each iteration's potentially shippable increment.

The phrase used in agile development for such an iterative, incremental process is "sashimi", a thin slice of a product which contains all aspects of the final product. The word sashimi is related to sushi; one slice of sushi is similar to all other slices.

A development team must have excellent engineering practices and tools to consistently develop a truly potentially shippable product increment. See Phase 4: Development Environment, for evaluating and staging these practices, as required.

Phase Name: ***2 Define Increment of Shippable Product***

Path	Type	Characteristics
2A	Internal software development	All testing and documentation
2B	Commercial software development	All testing, documentation, user manuals, and distribution
2C	FDA life critical	FDA documentation and requirements traceability
2D	Mission critical	Requirements traceability
2E	Package selection	Package selection criteria

Path Name: *2A Internal Software Development*

Beyond internally and externally tested, working code, this includes all artifacts such as documentation, traceability, models, and simulations.

If all artifacts and documentation required by this organization haven't been fully defined and aren't well known to the development team, put the following item as a top priority Product Backlog:

"Define all documentation and artifacts that are part of each increment of product functionality"

Path Name: *2B Commercial Software Development*

Beyond internally and externally tested, working code, this includes all artifacts such as documentation, traceability, models, and simulations.

If all artifacts and documentation required by this organization haven't been fully defined and aren't well known to the development team, put the following item as a top priority Product Backlog:

"Define all documentation and artifacts that are part of each increment of product functionality"

Path Name: *2C FDA Life Critical Development*

Beyond internally and externally tested, working code, this includes all artifacts such as documentation, traceability, models, and simulations.

If all artifacts and documentation required by this organization haven't been fully defined and aren't well known to the development team, put the following item as a top priority Product Backlog:

"Define all documentation and artifacts that are part of each increment of product functionality"

Path Name: *2D Mission Critical Development*

Beyond internally and externally tested, working code, this includes all artifacts such as documentation, traceability, models, and simulations.

If all artifacts and documentation required by this organization haven't been fully defined and aren't well known to the development team, put the following item as a top priority Product Backlog:

"Define all documentation and artifacts that are part of each increment of product functionality"

Path Name: *2E Package Selection Development*

Beyond internally and externally tested, working code, this includes all artifacts such as documentation, traceability, models, and simulations.

If all artifacts and documentation required by this organization haven't been fully defined and aren't well known to the development team, put the following item as a top priority Product Backlog:

"Define all documentation and artifacts that are part of each increment of product functionality"

Phase Name: *3 Multi-team or Offshore Development*

Some project require a systems and product architecture to be established. If this project is one of those listed in the below path map, it is recommended that you establish high priority product backlog to ensure that the first Sprint develops such architectures, in addition to the normal potentially shippable product increment.

Andy Hunt in *The Pragmatic Programmer* provides guidance for how to build such an architecture with working functionality. He calls it a "tracer bullet" approach to incremental development.

The situations where it is recommended that you establish such architectures are identified below:

Path	Type	Characteristics
3A	Single team Development	A single optimized team will develop the software
3B	Multi-team Development	More than one team will be working on the project
3C	Offshore Development	Some or all of the work on the project will be done offshore.

A common and unfounded observation about agile processes is that they don't scale. This belief springs from the early application of agile processes to small projects and the articles describing these projects.

An agile project described in the literature is optimized for productivity. It often consists of one small team of experts that is located together in an open space using pair programming with immediate access to the customer, potential users, and decision makers – and a bunch of other optimizing practices. Jim Coplien studied the Quattro for Windows (BWP) project at Borland in the early 1990's. The project employed many of what came to be known as agile practices. Indeed, Jim's writings[1] about the project were the genesis of some of the agile processes. Jim documented the following observations comparing what he saw at Borland to industry averages:

```
1,000,000 lines of C++ code.........BWP.........Industry standard
Time in months.....................31..........>50
Staff..............................8...........>100
Function points per staff month......77..........2
```

(James Coplien. Borland Software Craftsmanship: A New Look at Process, Quality, and Productivity. Proceedings of the 5th Annual Borland International Conference, Orlando, 1994.)

Subject:3 Multi-team or Offshore Development

This type of radical productivity can lead one to extrapolate that a project normally staffed by 500 people could be done with a team of 40 using agile processes. However, as Martin Fowler observed at the 2003 Canadian Workshop on Scaling (Canadian Workshop on Scaling XP/Agile Processes, Banff, Alberta, Canada, February, 2003)., "scaling agile projects is the ***last thing*** you should do." This doesn't mean that you can't scale an agile project. It means that you should first do everything you can not to scale one. Because, in scaling the agile project, you start to diminish the effectiveness of the very practices that produce such radical productivity.

At this workshop, the question was posed as "how do you scale agile projects?" This question arises from normal management logic of "If we have a project that we estimate at 1,000 function points in size and we want it done in 10 months and our staff average productivity is 2 function points per month, we will need to assign 50 people to the project to get it done in time."

We flipped the question to read, "how do you take any large project and reduce it to a small, agile project?" In light of the productivity numbers quoted by Jim Coplien, above, and similar productivity experienced repeatedly within the agile community, this seems like a more cost-effective and prudent course of action. For instance, let's use a modest agile productivity multiplier of a factor of 10. Recalculating the example project parameters, the agile answer is "*If we have a project that we estimate* at *1,000 function points in size and we want it done in 10 months and our staff average productivity is 20 function points per month, we will need to assign 5 people to the project to get it done in time."*

Of course, agility can't just be declared. A lot of work has to be directed by management to produce an agile project. Management is going to have to be very proactive, getting experts, getting everyone together, providing the best tools, using agile engineering and management practices, and changing development approaches. But the payback seems very attractive.

ThoughtWorks is a software development company headquartered in Chicago that specializes in agile software development, using excellent professionals who are well trained and well equipped. ThoughtWorks charges a premium for undertaking projects, but the type of productivity indicated above might mean that any reasonable premium is a bargain.

I have managed projects of a very large scale using agile processes. In these cases, reducing the number of people through productivity improvements wasn't an option. I used the Scrum agile process to manage these projects, taking advantage of the coordinating mechanisms and outstanding visibility into project progress and problems provided by Scrum.

The worst such project was a Y2K project at a healthcare software provider. Previous attempts to plan, coordinate, and manage this project using traditional project management techniques had failed. Scaling is often thought of in terms of number of people involved in a project, since that often imposes the greatest complexity. The Y2K project had over eight hundred people involved. The project was also very complex in a number of other dimensions usually not found together in a single project. These scaling dimensions include the number of organizations involved (350+), the application (delivery of health care), the criticality (mission critical), the distribution (multiple departments, multiple organizations throughout North America), the type of software (interpretive mainframe, client-server, and web), the type of software work involved (new development, enhancement, maintenance, and bug fixing), the size of the software (over 2500 function points), and the length of the project (over two years).[3]

Subject: 3 Multi-team or Offshore Development

More of my customers have been asking me how to use agile processes, particularly Scrum, to help them manage offshore development. Since offshore development undercuts many of the practices that promote agile productivity, I ask them why they don't just increase the productivity of their teams by thoroughly introducing agility? It seems that offshore development, with its potential for lower unit costs (dollars per programmer day), offers management hope that their losses can be reduced. Since the project is probably going to fail anyway, let's minimize our losses by lowering our investment by using lower priced resources. A more optimistic, agile, way of looking at this problem is to fix the problem at home and increase the probability of success.

The agile process "sweet spot" occurs with teams of seven people, give or take two. These teams can be extraordinarily productive, measurements indicating a potential increase of productivity at least 35 times more than average. Many inadvertent practices reduce this productivity, including scaling, so let's understand how to be as productive as possible before we introduce scaling – which reduces team productivity for such goals as quicker time to market.

High bandwidth communication is one of the core practices of agile processes. If a team has more than seven people, they tend to need to revert to written documents and formal models to keep everyone's activities synchronized. The best communication is face to face, with communications occurring through facial expression, body language, intonation, and words. When a white board is thrown in and the teams work out a design as a group, the communication bandwidth absolutely sizzles.

Until the late 1990's, many of the engineering practices have promoted formal documentation of communications, such as formal models, documentation templates, and computer aided software engineering tools. Every time I don't work directly with team members using face to face communications, however, I reduce the communication bandwidth and introduce the probability of misunderstandings. When I write something, I'm trying to formulate ideas, understandings, and experiences into words. When you read my writings, you try to understand what I'm saying within the context of your experiences and context. In the process of narrowing my bandwidth to words, and you trying to expand the bandwidth from words to your understanding, a lot is lost. No matter how well I write and you read. And, most of us are not superb writers and readers.

Many agile practices are aimed at maximizing communication bandwidth. These include:

1. Only using modeling tools and techniques to guide thought processes while on the path to code. Models are not used to document, but to ensure the rigor of the thought process.
2. Collocating teams so that any team member can readily get face to face with any other team members to talk and diagram through a problem.
3. Collocating teams in open spaces to maximize the access within the team. If I want to ask a fellow team member something and I get up from my office, go down the hall, look in his or her office, and find that they aren't there, I have both wasted time and lost the thread and depth of my thinking. More than just time was wasted.
4. Collocating teams in open spaces so team members can see each other, see how each other is doing and feeling, and hear when a conversation is occurring in which they want to participate. Privacy is easily obtained by putting on headphones from Bose.
5. Keep iterations to thirty calendar days. Longer iterations require communications persistence through such artificial techniques as documentation or modeling tools. If the time between learning a requirement to finishing tested code is kept to under thirty days, the problem and its solution can usually be kept in the mind.

6. Keep the team size as close to seven as possible. Seven minds seem able to attain and maintain a shared mental model of a system and its representation in design and code without artificial aides such as documentation. Misunderstanding and recording time is minimized.
7. Use a shared code library and rigorous coding standards so everyone on the team can readily read and understand the system. If modeling documentation is minimized, the code literally is the design. The code must be easy to read and unambiguous. Variable naming is just one example of these standards.
8. Use agile engineering practices so the team always knows the status of development. Test first development ensures that the code reflects the design and that the code is tested as soon as possible. Source code management, continuous integration, and automated testing suites find errors as quickly as possible. Refactoring keeps the design simple, elegant, and easy to debug. Not writing arcane, heroic algorithms keeps code easy to understand. All of these practices combined mean that when I think I have a working system, it really is a working system that is sustainable and maintainable. This is called an increment of potentially shippable (implementable) product functionality.
9. Hold short daily status meetings. Face to face, team members communicate status and problems with each other. At full bandwidth, the team synchronizes itself daily.

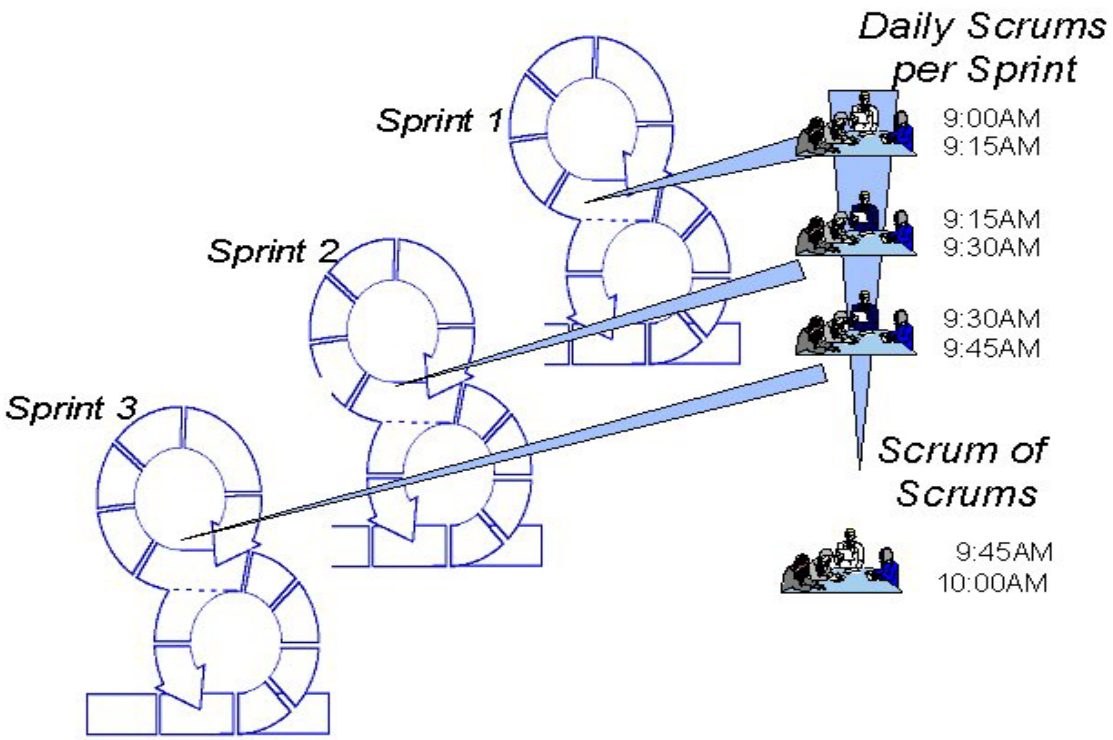
Every scaling practice will reduce the productivity of these teams in support of other goals. Our job will be to understand how to implement these scaling practices as intelligently as possible, so we don’t throw out the baby with the bath water.

Subject: 3 Multi-team or Offshore Development

When it is necessary to use more than one team, each team should select from the Product Backlog and work on areas that are as highly cohesive as possible and as loosely coupled to any work other teams are doing as possible. That is, each team's work should be as independent of any other teams work as possible. The coordination required should be minimized.

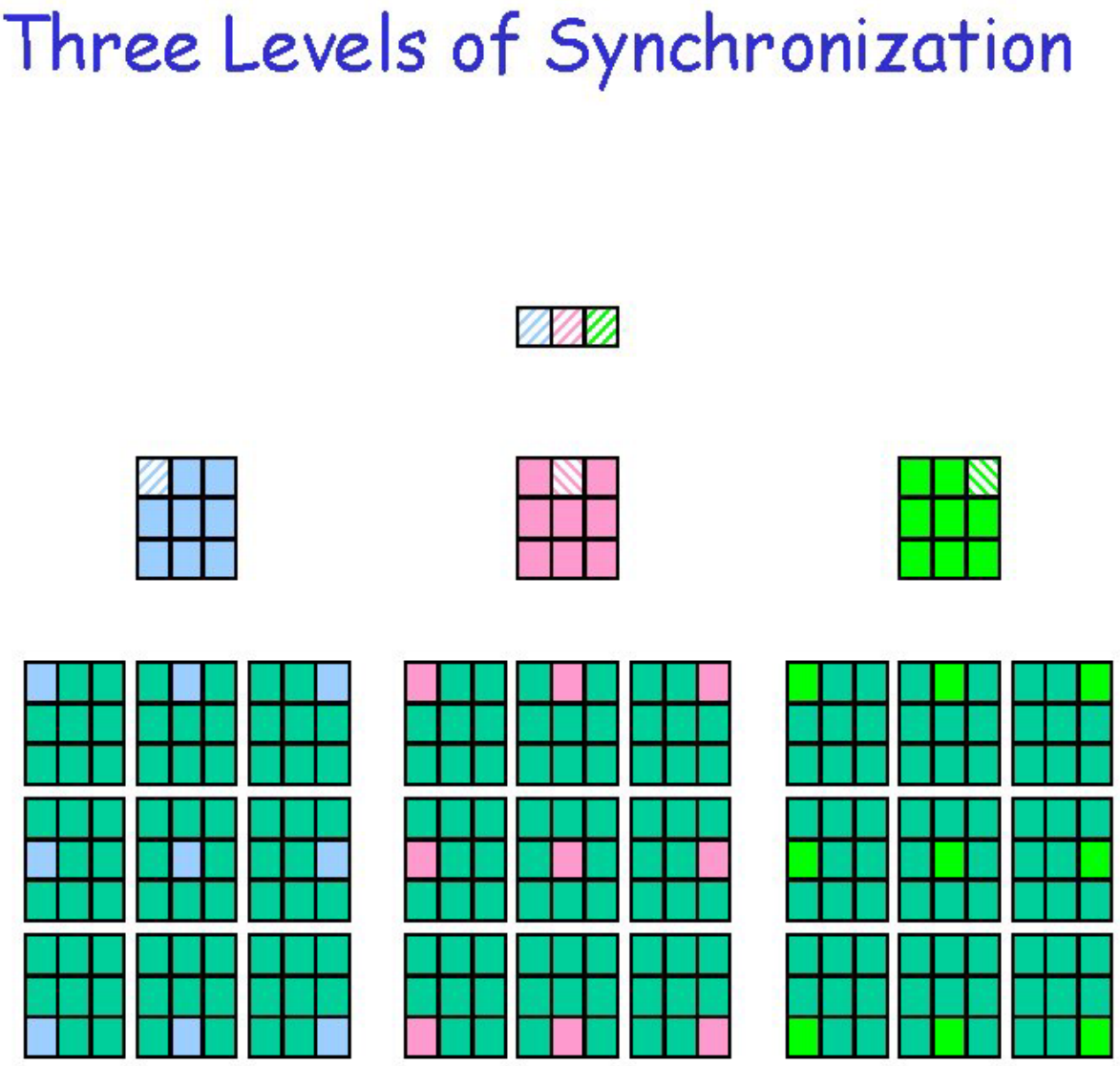
As the teams begin their Sprints, each team coordinates its own work with Daily Scrums. If multiple teams are utilized, their work should be coordinated also by Daily Scrums, sometimes called Scrum of Scrums. These higher level coordinating Daily Scrums are held after the lower level Daily Scrums, are attended by one member of each of the teams to be coordinated, and operate exactly as any Daily Scrum - except that teams rather than team members are being coordinated.

In some projects, several coordinating levels of Daily Scrums are required. When this is necessary, the frequency of coordination can diminish in the higher level Daily Scrums (or bi-Daily Scrums, or Weekly Scrums), because the level of interaction diminishes.



Subject: 3 Multi-team or Offshore Development

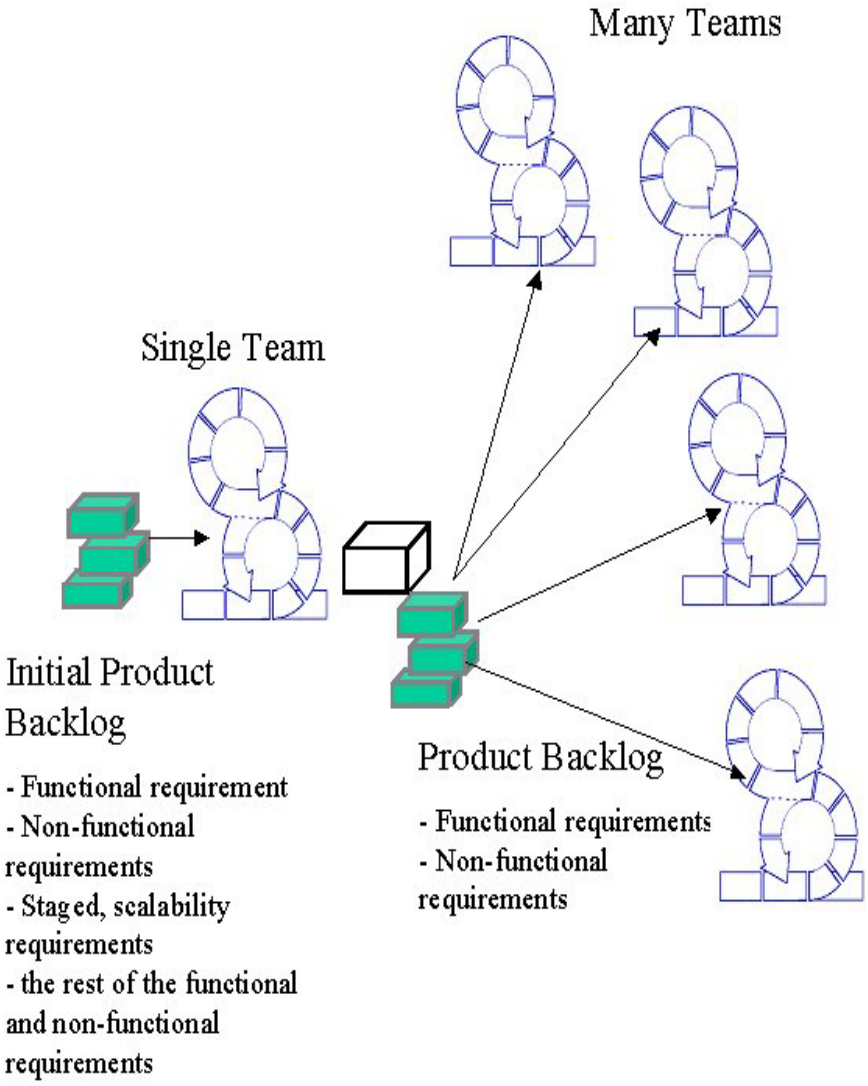
An example of three levels of synchronization was provided courtesy of Mike Cohn of Mountain Goat Software, copyright 2003.



Reproduced with permission from Mike Cohn,
Mountain Goat Software, 2003

Subject: 3 Multi-team or Offshore Development

When multiple teams are used, the project is still started by using one team, as follows:
1. The Product Backlog is constructed so that important architectural, environmental, and infrastructure elements are at the top of the Product Backlog - along with some user functionality that will demonstrate the architecture in operation.
2. A single team starts work on the Product Backlog and persists until a stable enough architecture, environment, and infrastructure is in place for more than one team to begin operation.
3. The initial team is broken up and used to seed the multiple teams that begin Sprinting.



4. The Business Architecture is divided into packets of functionality that are highly cohesive and have as little coupling between each other as possible. Each packet is named for its primary functionality.
5. The Product Backlog has another column added to it for describing the business functionality packet to which each Product Backlog item belongs.

Each of the multiple teams chooses work for their Sprints from the Product Backlog that is as orthogonal to that of other teams as possible, that is, Product Backlog from the same packet of business functionality.
6. The teams begin Sprinting, coordinating the work within teams using the standard Daily Scrum..
7. Coordinating Scrum of Scrums are established to synchronize the work of the multiple teams.

Priority (1-9)	Function	Full Description	Raw Dev. Effort
1	General	Setup development environment	4
1	General	Confirm use of zope as development environment	2
1	Membership	Ability to sign up for various level of membership	4
1	Membership	Ability to use credit card to pay for membership	15
1	Membership	Provide extract from database to external sources	3
1	Membership	Notify members with membership data	5
1	Membership	Generate receipts and certificates	5
1	Membership	Tie membership program to bank accounts	5
1	Membership	Implement open access database (MySQL?)	6
2	News	Authoring environment for news	8
1	Website	Website look, feel, navigation, initial pages	10
3	Founders Page	Include agilealliance.org founders page	10
3	Sponsors	Display sponsors and link to web sites	6
4	Articles	Authoring environment for articles	10
4	Articles	Organizing and sorting capability for articles	4
4	Articles	Library catalog for articles	8

Path Name: *3A Single Team Development*

All requirements for single team development are defined in Phase 4: Development Environment. Single teams are recommended as their productivity is higher than any other development structure.

Path Name: *3B Multi-Team Development*

Multiple teams require mechanisms to coordinate their work. Dependencies between the teams must be minimized and the teams require a shared work environment wherein ongoing builds and tests can be employed to test the completeness and stability of the system. In general, the mechanism for coordinating the work of multiple teams reduces overall productivity of each team.

The mechanisms recommended for coordinating teams are:
- A business architecture that is parsed into subsystems with high cohesion and low coupling. Enough subsystems should be defined to support the number of teams envisioned for the project.
- A systems architecture that will further guide the division of business subsystems.
- A development environment that supports multiple teams that aren't collocated.

The process for developing and using these multi-team mechanism is:
1. Define Product Backlog indicating that such work has to be done. Make it high priority within the Product Backlog.
2. Form one development team of people skilled in architecture, design, environments, and working with teams.
3. This one development team will be the only development team that initially works on the project. They will Sprint until a solid infrastructure that supports multi-team development is in place. However, while it is developing such an infrastructure, the team is required to deliver useful, meaningful business functionality every Sprint. Even though such functionality may be minimal, as described by the phrase "tracer bullet,", it is required every Sprint.
4. When the infrastructure is in place, deploy multiple teams. The initial team is used to seed these teams, continuing to be "pigs," committed to the develop of user functionality within the teams that will continue to Sprint.

Activity ID: *3B.1 Develop Business Architecture*

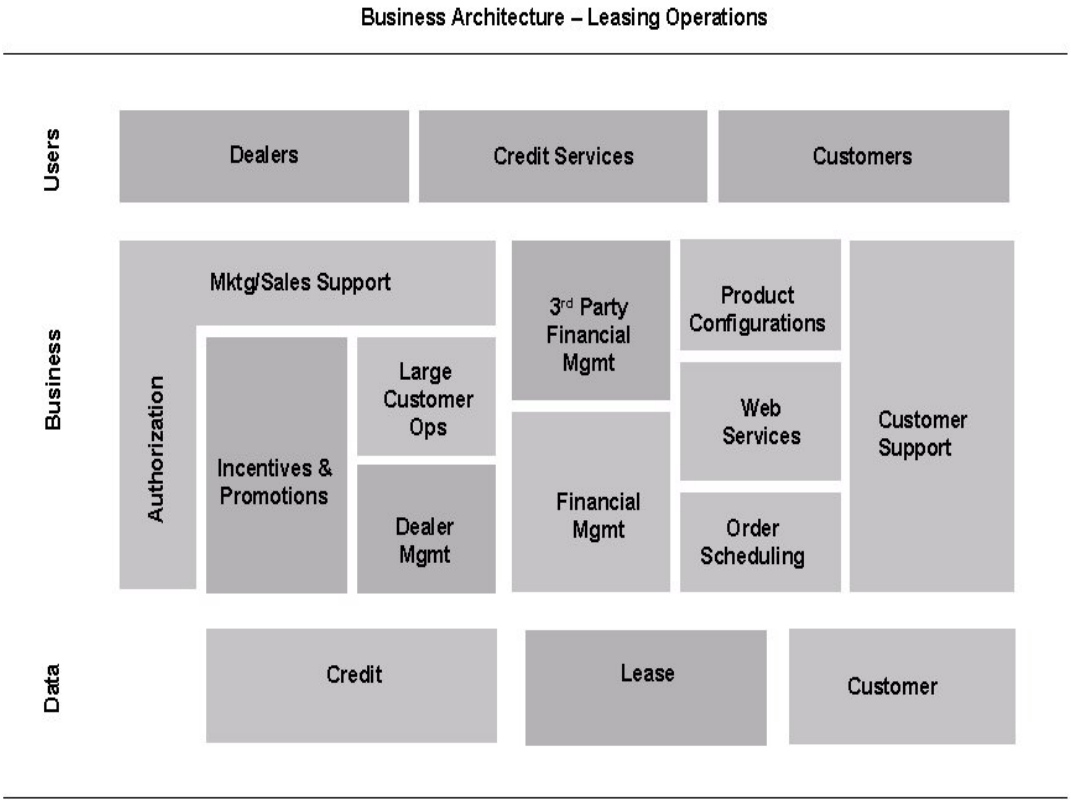
Responsible: *Product Owner, Team*

Purpose: The architecture of the system as it will be implemented within the business operation for the customers and users must be developed in the early Sprints prior to multiple teams being employed. Only one team should be active when the architecture is designed. . Ensure that this has been done to a level of detail that supports parsing the functionality among more than one development team with minimum coupling between teams and the highest degree of cohesion within the functionality.

Assumptions: The system will be developed by the teams iteratively, sashimi style, and they require the ability to parse the business functionality among many teams.

Risks: Any imprecision will lead to conflict and dependencies between development teams that will require the intervention and attention of the Product Owner and Team to resolve.

Description: Add to the Product Backlog as a high priority item:
"Detail Business Architecture", to decompose the model that was built in the Planning phase to an adequate level of detail for parsing the work amongst teams.



Activity ID: ***3B.2 Develop Systems Architecture***

Responsible: ***Product Owner, Team***

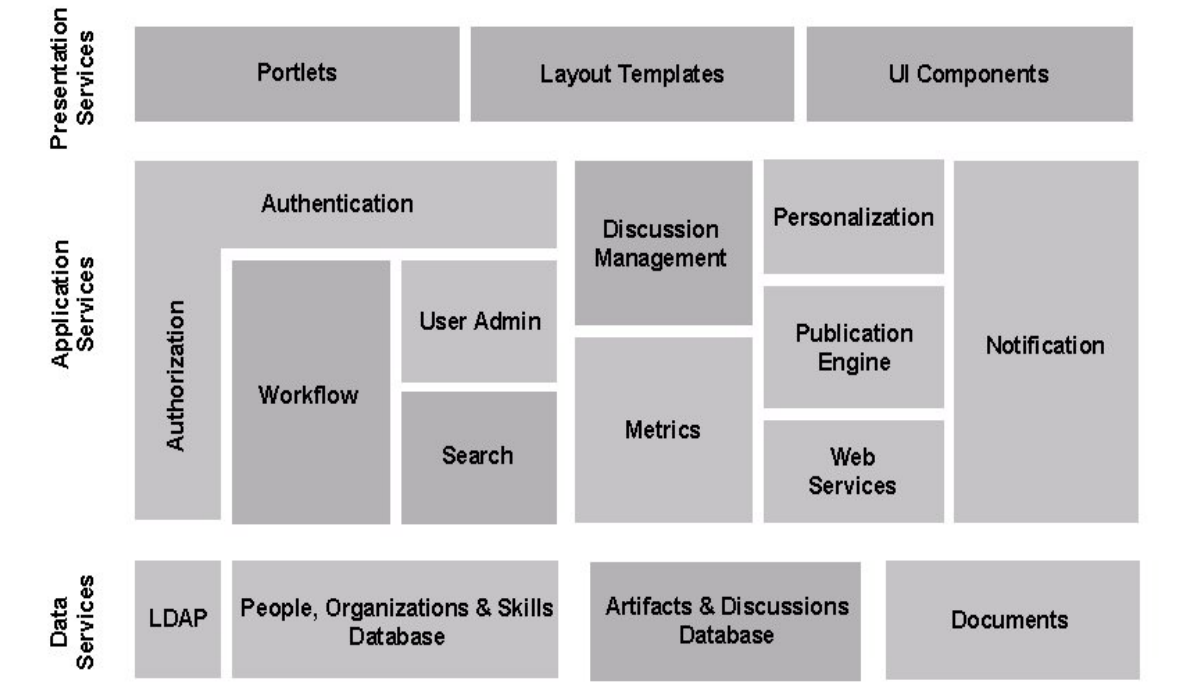
Purpose: The technical architecture within which the business architecture fits and within which the business functionality will be delivered must be developed in the initial Sprints prior to the addition of multiple teams. One team should be initially employed to develop such an architecture until it has enough detail to accurately parse the work.. This architecture spells out all interdependencies, technology, and technologies for implementing the business system, both at the quality assurance and implementation details.

Assumptions: The Development Team fully tests the selected technologies to ensure that the system can be implemented using them.

Risks: If any of the technologies don't work as expected and don't support their part of the architecture, the Development Team must promptly identify this during the development Sprints and assist the offshore development teams in a timely resolution.

Description: Add to the Product Backlog as a high priority item:
"Detail Systems Architecture" to decompose the model that was built in the Planning phase to an adequate level of detail for parsing the work amongst teams.

Systems Architecture



Activity ID: *3B.3 Define Development Environment*

Responsible: *Team, ScrumMaster*

Purpose: A development environment that will be used by the multiple teams that aren't located geographically together and may work in different time zones must be defined and implemented prior to multiple teams being assigned to the project. Such development should be done by the initial team during the beginning Sprints of the project. Phase 4 addresses the definition of a development environment. Such an environment must be defined so that it scales to multi-site, multi-team, multi-time zone development.

Assumptions: The project is going to require more difficult to use and more difficult to learn tools to generate such development coordination. One or more infrastructure people may be needed to be added to the teams to develop and maintain such an environment.

Risks: Without an adequate multi-team environment, the communications bandwidth between the various teams may be seriously compromised.

Description: Add to the Product Backlog as a high priority item:
"Design Multi-team development environment"

Path Name: *3C Offshore Software Development*

Offshore development benefits from the frequent inspection and adaptation provided by Scrum. There is an opportunity for this at the end of the Sprint review. There is also an opportunity for this at each Daily Scrum. However, distances and differences in time zones may make such coordination difficult. Regardless, it provides the only benefit afforded by Scrum to offshore development, so every effort should be made to comply.

Offshore development violates almost every other Scrum practices that provides the high productivity and quality. This isn't unique to Scrum, but is a problem for any development process. For instance, Scrum utilizes incremental development, with each Sprint developing a complete slice of product functionality - from requirements through design to coding and testing. Offshore development requires the development of requirements and architecture at the customer site, and the detailed design, testing and coding at the offshore site. Then acceptance testing and the round of bug fixes and change orders takes place. The customer must fully define all of the requirements upfront, building an inventory that may go obsolete as business requirements change. While the offshore developers design and code the application, it the functionality also may go obsolete and be unneeded.

Another tenet of Scrum that offshore development violates is the ability for the customer to steer the project Sprint by Sprint, based on inspection of each iterations working functionality. The customer ensures that the top priority functionality is developed first, and may not even have lower priority functionality developed. Without this frequent collaboration between development teams and customers, much that the customer doesn't require may be build regardless, and that which is built may not deliver the top business value.

Still another violation of Scrum productivity practices is that absence of full bandwidth communication between all team members, ensuring that nuances that are difficult to capture in documentation are captured. The moment communication occurs through documentation and models, the change for error occurs. The larger or more complex the project, the greater the chance.

Despite all of this, if offshore development is in the cards, employ the following activities before entering into Sprints. Then use the Sprint mechanism to monitor and guide the offshore development teams.

Activity ID: ***3C.1 Develop Requirements***

Responsible: ***Product Owner, Team***

Purpose: Based on the vision, develop a full set of requirements, both functional and nonfunctional for the envisioned system or product. Using modeling techniques, such as UML, fully describe all of the entities of the system, their methods, their sequences, and their interactions.

Assumptions: The team has all of the domain knowledge with itself or available to it to fully model the expected system in every detail. No decision remain unmade or ambiguities exist.

Risks: Any imprecision in the requirements document will result in erroneous functionality that can only be caught through extensive user acceptance testing, as well as extensive analysis of the resulting data.

Description: For every requirement, create an item in the Product Backlog. Add another column to the product backlog the can be used to point the various documents and models that fully spell out the definition of the requirements.

Activity ID: ***3C.2 Develop Business Architecture***

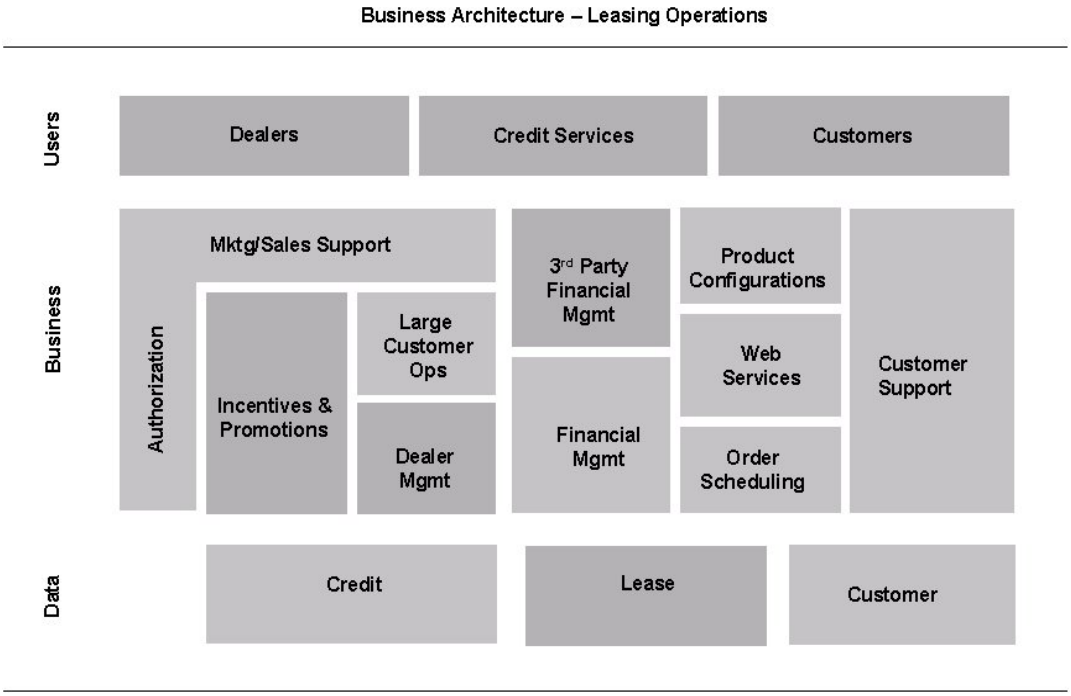
Responsible: ***Product Owner, Team***

Purpose: Develop the architecture of the system as it will be implemented within the business operation for the customers and users. Ensure that this has been done to a level of detail that supports parsing the functionality among more than one development team with minimum coupling between teams and the highest degree of cohesion within the functionality.

Assumptions: The system will be developed by the offshore developers iteratively, sashimi style, and they require the ability to parse the business functionality among many teams.

Risks: Any imprecision will lead to conflict and dependencies between offshore development teams that will require the intervention and attention of the Product Owner and Team to resolve.

Description: Fully decompose and detail the model that was built in the Planning phase to its lowest level of detail. This model will help explain the overview and intricacies of the business domain to all parties..



Activity ID: *3C.3 Develop Systems Architecture*

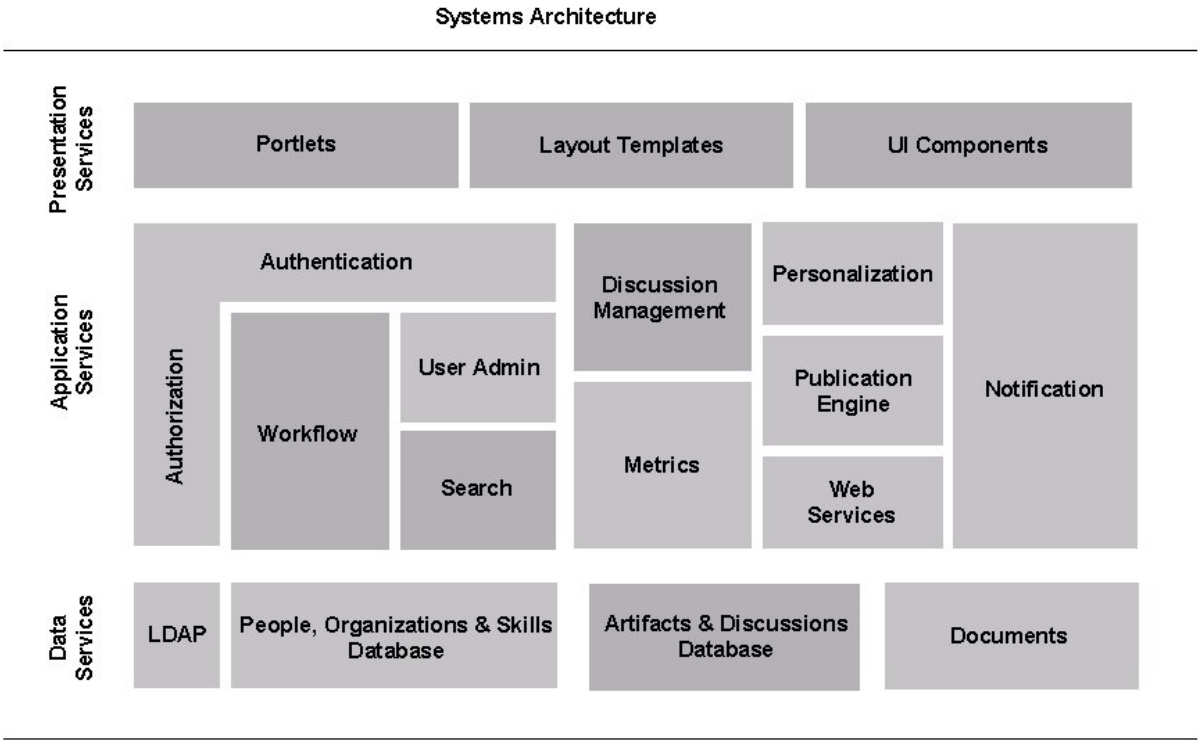
Responsible: *Product Owner, Team*

Purpose: Develop the technical architecture within which the business architecture fits and within which the business functionality will be delivered. This architecture spells out all interdependencies, technology, and technologies for implementing the business system, both at the quality assurance and implementation details.

Assumptions: The Development Team fully tests the selected technologies to ensure that the system can be implemented using them.

Risks: If any of the technologies don't work as expected and don't support their part of the architecture, the Development Team must promptly identify this during the development Sprints and assist the offshore development teams in a timely resolution.

Description: Fully decompose and detail the model that was built in the Planning phase to its lowest level of detail. This model will help explain the overview and intricacies of the system's architecture to all parties..



Activity ID: *3C.4 Define Development Environment*

Responsible: *Team, ScrumMaster*

Purpose: Define a development environment that will be used by the multiple teams that aren't located geographically together and may work in different time zones. Such a development environment must support source library sharing across all teams. Phase 4 addresses the definition of a development environment. Such an environment must be defined so that it scales to multi-site, multi-team, multi-time zone development.

Assumptions: The project is going to require more difficult to use and more difficult to learn tools to generate such development coordination. One or more infrastructure people may be needed to be added to the teams to develop and maintain such an environment.

Risks: Without an adequate multi-team environment, the communications bandwidth between the various teams may be seriously compromised.

Description:

Activity ID: *3C.5 Develop Acceptance Tests*

Responsible: ***Product Owner, Team***

Purpose: Since the teams that developed the requirements are different from the teams performing the design, coding and unit testing, a way must be present to ensure that the requirements have been fully complied with through working, solid functionality. Acceptance tests automated into a testing tool provide such a mechanism.

Assumptions: If the team has developed Use Cases to describe the user interaction with the anticipated system, these should be the basis for defined rigorous acceptance tests.

Risks: To the extent that these tests are incomplete, there is no mechanism that ensures the presence and solidity of the expected functionality.

Description:

Phase Name: *4 Development Environment*

As a team starts working together, one of the first tasks is to ensure that they have a solid, mutually shared set of engineering practices, methods, and tools. The first several Sprints may have backlog embedded into them to progressively build such an environment. The below table identifies important aspects of the development and the backlog items that should be placed into the product backlog if such aspects aren't present.

Experts develop better systems. This seemingly trite expression has been misapplied to agile processes, where some have alluded that agile only works when done by teams of experts.

Agile processes don't require experts. Agile teams can be composed of people with any set of software engineering skills. However, the practice of frequent inspection confronts management with the consequences much sooner than traditional prescriptive development processes. Every day, the consequences are seen at the daily Scrum. Team members report that the build didn't work, that their code was overwritten, that the components don't match the methods in the class models. At the Sprint review, the team presents its increment of working functionality. Except, the functionality is barely demonstrable because of bugs. Or, the user interface is so poorly constructed that the customers start objecting. Or, in subsequent iterations the productivity declines because of poor design and inadequate refactoring. The bad news of inadequate team staffing is apparent immediately, whereas traditional projects usually hide the bad news until the end of the project, when the system isn't ready.

When I first implemented Scrum, the engineering teams and engineering practices were pretty solid and work progressed smoothly. As agile practices have spread, a wider audience of organizations and skill sets are attempting to use agile practices. Some of the more dramatic failures have caused the observation that "experts are needed." To the contrary, I feel that the problems highlighted by agile processes are valuable. They indicate to an alert and intelligent management the exact engineering improvements that are needed. Remediation plans can be drawn up and action taken. The actions, such as training or bringing in expert consultants as mentors, can be included in the iterations. The engineering practices and resulting increments of functionality improve, Sprint by Sprint, until a solid team and product is realized.

Subject: 4 Development Environment

An advantage of Scrum over many processes is it can be implemented in one day. A disadvantage of Extreme Programming is the time needed to implement its full set of practices. However, when I implement Scrum and find inadequate engineering practices, I've started to implement Extreme Programming also. As the organization gets the benefits of iterative, incremental development, it's engineering practices also incrementally improve.

An increment of potentially shippable product functionality is demonstrated at the end of every Sprint. I've found that specific engineering practices are necessary for this to occur, much less start to realize the productivity inherent in agile processes.

Source code management
Inadequate source code management is easily detected through overwritten and lost code. Check in/check out, version control, branching, and release management are all necessary. If the team is not collocated, or multiple teams are involved, a multi-site source code management system is required.

If this capability is not in place, insert into the Product Backlog at a high priority:
"Nonfunctional Requirement - Investigate and implement source code management"

Test driven development
This ensures that tests fully reflect the functionality and visa versa. This practices causes the developer to think through the design through thinking through the test, all prior to coding. Use either this or other unit testing practices to ensure a full suite of unit tests.

If unit testing is not a standard practice employed by the team, insert into the Product Backlog at a high priority:
"Nonfunctional Requirement - Learn and implement test driven development"

Automated builds
Every time code is checked in, build the software. If the build fails, fix the underlying problem. Otherwise, the problems just accumulate until nothing can be demonstrated at the end of the Sprint. Testing suites are aggregations of unit, string, system, and stress tests. These are applied in the small after every build, on the average at the end of every day, and in total at least weekly. Without this testing regimen, the quality, internal integrity, external integrity, and viability of the functionality cannot be known.

If this capability is not in place, insert into the Product Backlog at a high priority:
"Nonfunctional Requirement - Investigate and implement automated build and test capability"

Refactoring
As functionality gets added to code, redundancies, inefficiencies, and awkward structures occur. For example, the same core functionality may be expressed in multiple instances, each instance containing small variations from the other. The practice of refactoring means that the engineers ruthlessly remove these inefficiencies when they are found. Time is allocated for the code to be cleaned up. Management and the engineers know that short term gains from accepting sloppy code is more than

paid for later in the project.

If this capability is not in place, insert into the Product Backlog at a high priority:
"Nonfunctional Requirement - Investigate, learn and implement refactoring"

Coding Standards

Coding standards ensure that the code is uniform and readable.

If this capability is not in place, insert into the Product Backlog at a high priority:
"Nonfunctional Requirement - Devise and implement coding standards"

User Development of Acceptance Tests
Users understand what the functionality should do. They should develop tests to assure themselves that the software delivers the functionality as expected. Since Scrum delivers functionality at the end of every Sprint, customers should be developing acceptance tests throughout the Sprint and include them as part of the automated build process.

If this capability is not in place, insert into the Product Backlog at a high priority:
"Nonfunctional Requirement - Investigate and implement user development of acceptance tests"

Frequent Check-in of Code
Code needs to be integrated and built frequently to ensure that it fits together. Frequent, at least daily, checking in of code is a practice that provides this assurance.

If this capability is not in place, insert into the Product Backlog at a high priority:
"Nonfunctional Requirement - Set standards for checking in code"

Shared Code
Code should not be developed in isolation. Such practices as paired programming and code reviews help to ensure the quality of the code and also provide an opportunity for learning. The team should share the entire code library to ensure the consistency, quality, and standardization of code.

If this capability is not in place, insert into the Product Backlog at a high priority:
"Nonfunctional Requirement - Investigate and implement code review and sharing practices"

Working Environment

Communication bandwidth between team members needs to be maximized in Scrum. Maximized communication in small teams takes the place of extensive documentation. To maximize bandwidth, the team should be provided with:

- a team work space where they can meet as required to discuss design, issues, and any outstanding development decisions. This room should be equipped with a speakerphone, flip charts, and more than one whiteboard. This room may double as the room where the Daily Scrum is conducted.

Subject: 4 Development Environment

- a working environment where team members can readily see and hear each other. If cubicles are currently being used, minimize them. Remove walls. Put a shared work space, such as tables, in the middle of the opened up cubicles.

- More than sufficient servers, networks, workstations and software. If any of these are substandard, they impede the developers and reduce productivity.

If this capability is not in place, insert into the Product Backlog at a high priority:
“Nonfunctional Requirement - Upgrade working environment and tools for teams”

These and other practices aim toward one thing: the increment of product functionality demonstrated at the end of each Sprint has to be potentially shippable. If these practices aren’t employed, one just doesn’t know how complete the functionality is and how much work remains. Some organizations think that they can get around the need for shippable increments. They organize a separate quality assurance group that tests the code afterwards. This practice ensures that completeness can’t be measured every Sprint. To address this problem, integrate the testing engineering with the development engineering on each team. The teams are intended to be cross functional, able to address all of the aspects of engineering the demonstrable functionality.

Phase Name: *5 Project Staffing*

The business project manager and IT project manager are jointly responsible for staffing the project with development and business domain knowledge and skills. Each project team is of limited size, with a maximum of ten to twelve staff members. Teams that are larger than this require a communications infrastructure that reduces their productivity and hinders their ability to self-organize. Such teams should be divided to create more than one team.

A Scrum team is cross-functional, with all of the skills within it needed to build an increment of product functionality. The tasks the team performs includes planning an Sprint, analyzing the functionality, designing a solution, building the solutions and any documentation, and testing that the solution works as expected. The skills in the team will vary from Sprint to Sprint, and span business and technology domain knowledge and skills, analysis skills, design skills, development skills, documentation skills, and testing skills. Most of the people on the project are full-time, but some people may be assigned to the project on a part-time basis.

The best people available should be assigned to the team. The team will produce increments of functionality as best it can; the better the team's skills, the more functionality it can produce. Scrum maximizes people's ability to employ their skills, but is constrained by the skills that are applied. The business project manager ensures the team has adequate business domain knowledge; this maximizes the accuracy of the functionality produced and minimizes the time that developers spend waiting for decisions. The IT project manager ensures that adequate IT skills are available to maximize the utility of the team members with business knowledge.

Teams self-organize. That is, the team is responsible for collaborating among itself to figure out how to build an increment of functionality within the Sprint, and to figure out who will do what work. Some team members are more senior and experienced; they exert more influence and are natural leaders within the team. However, no one is in charge. Instead, the team members are collectively responsible for figuring out how they will deliver the business functionality. This may sound like chaos, but given the pressure of the commitment to create functionality within the Sprint, the team quickly focuses on the work, arranging itself as best as possible.

The project should be staffed with the best people available, including people with:

- user, customer, or business domain expertise;
- technology domain expertise;
- testing domain expertise;
- documentation domain expertise (if documentation is to be build during the Sprint); and,
- infrastructure, build, and source code management expertise.

Maximum productivity is derived from experts. However, if cross-training or mentoring is desired, the staff can be mixed between experts and novices. Ensure that the team knows that management has taken this loss of productivity into account.

If the project will consist of more than one team, only staff one team for the first several Sprints. After the initial standards, architecture, and facilities are in place, seed additional teams from this team.

.

Phase Name: *6 Project Initiation*

Projects are started with an initiating workshop. If the team has never used Scrum, this workshop usually lasts three days. During the workshop the principles, practices and flow of a Scrum project are presented. They are then applied during the workshop to the project. By the end of the workshop, the project is underway using Scrum. This is possible because Scrum minimizes upfront planning, instead relying on self-organization of the teams and work, and emergence of the product and requirements.

The workshop flow is:

1. A trainer presents agile concepts, theory, practices, and an overview of the Scrum and Extreme Programming process and process flows.
2. The business project manager presents the business domain, the project definition and the requirements and release plan for the first year. The IT project manager then presents an overview of the product architecture, target technology, and development technology.
3. The trainer presents Product backlog, Sprint Planning, and Sprints concepts.
4. Team members introduce themselves. Since the team will be self-organizing, it's important that they get used to talking in front of each other. It is also important that they know each other's backgrounds, strengths, and perceived weaknesses.
5. The business project manager and the team define a product backlog with enough work to drive the team for several months of Sprints. They discuss the top five to ten requirements detail. The business project manager explains how they relate to the overall product.
6. The business project manager and the team brainstorm about how much functionality the team can build in the next Sprint. They scheme, plot, diagram, design and layout what the team believes it can build in the next thirty day Sprint. The Business Project Manager is there to validate their work and to answer any questions.
7. The team defines the Sprint backlog for the next Sprint to turn the selected product backlog into working functionality. The team self-organizes, defining what work it believes it will have to do to build the functionality and collectively deciding who will do what work. If a PERT chart is used to figure this work out, it is discarded immediately afterward.
8. The trainer presents the daily Scrum, end of Sprint, Sprint signature, and management topics. The team becomes aware of the daily status meetings. It finds out that management will attend daily to answer questions, make decisions, and remove anything slowing it down that is within management's power to remove.
9. The trainer presents engineering practices that will be followed during the project. The team becomes aware of continuous builds, test before coding, collective code ownership, paired programming, and other engineering practices it will use.

The project team starts the first Sprint. The team leaves the workshop and begins the actual work to turn the selected requirements into working functionality. If the team and organization already know Scrum, allocate one to two days and remove items 1, 3, 4, 8, and 9.

Phase Name: *7 Sprint Planning Meeting*

The purpose of the Sprint is to turn a set of the product backlog into an increment of potentially shippable product functionality. The product owner, ScrumMaster, and development team meet prior to every Sprint to determine what product functionality the team will work on. The product owner presents the product backlog and the team selects what it believes it can build during the Sprint.

Customers, management, users and other interested parties, also known as "stakeholders," are also welcome to this presentation. Regardless, the prioritization of the product backlog remains the exclusive responsibility of the product owner.

The Sprint planning meeting actually consists of two meetings. During the first meeting, the product backlog for the next Sprint is selected by the team. During the second meeting, the team identifies the Sprint backlog necessary to turn the product backlog into the increment of product functionality.

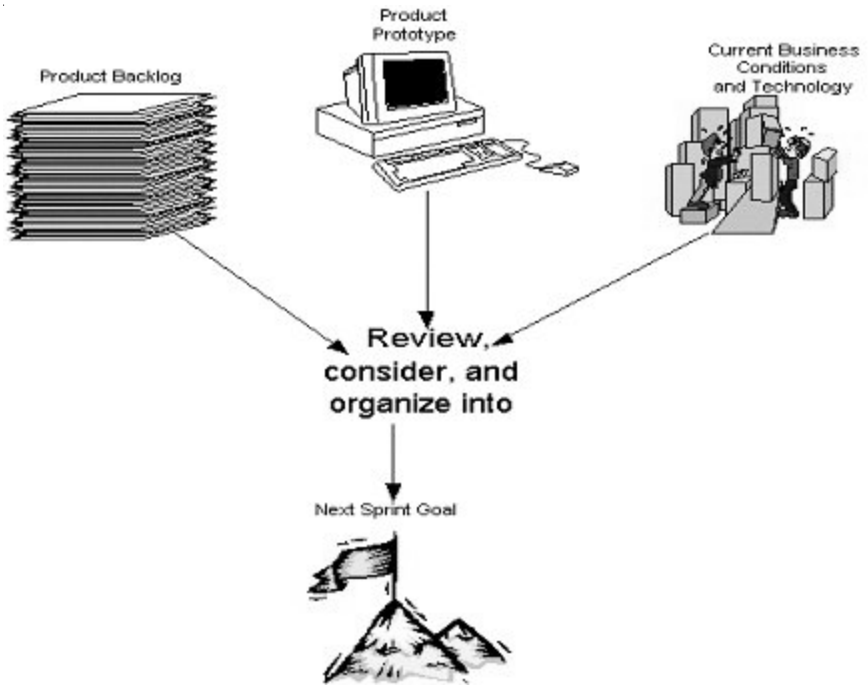
The first meeting lasts anywhere from 1 to 4 hours, depending upon how obvious the next set of work is.

The three inputs to this meeting are:

1. The previous product increment (unless this is the initial Sprint of the project);

2. The prioritized product backlog; and,

3. The current state of business conditions and technology.



The meeting starts by the product owner presenting the product backlog. If the product backlog has changed significantly since the last Sprint planning meeting, the product owner should discuss what caused the changes and their impact on the project. In presenting the product backlog, the product owner should focus on the top priority items, particularly that set of items which represents an estimated amount of work equivalent to what a team could maximally select to develop in one Sprint.

Subject: 7 Sprint Planning Meeting

The product owner and team collaborate about the meaning and details of these backlog items until the team has enough grasp of the intentions and details to arrive at their own estimate of how long it will take to turn the item into product functionality. This estimate supersedes the previous estimate for each item. The reason for this is that the team now has a better grasp of the technology and business domain than it did at the start of the last Sprint. To the extent that the product owner has been working with the team, frequently refining the estimates, this variation is minimal. Otherwise these variations may be a surprise.

The collaboration between team and product owner includes any input that any other participant in the Sprint planning meeting provides. The Sprint planning meeting is the one completely open meeting in Scrum, where everyone has a say. To the extent that the product owner has worked with everyone who has a stake in the product or system prior to the meeting, this input has already been gathered. To the extent that the organization is at odds about what should be done next, the Sprint planning meeting may have difficulty concluding. This is appropriate, ensuring that development is not initiated until organizational agreement is reached on funding the next Sprint. Since this agreement is required to start a Sprint, Scrum provides a thirty day checkpoint for how to best spend an organization's money.

Having selected the Product Backlog, a Sprint Goal is crafted. The Sprint Goal is the purpose of the selected functionality; that is, what should the selected backlog be able to demonstrate once it is built? The Sprint Goal is an objective that will be met through the implementation of the Product Backlog. For instance, this Sprint Goal could be:

Sprint Goal: to provide a standardized middleware mechanism for the identified customer service transactions to access backend databases.

The Sprint Goal gives the team some wiggle room regarding the functionality. For example, the goal for the above Sprint could also be: "Automate the client account modification functionality through a secure, recoverable transaction middleware capability." As the team works, it keeps this goal in mind. In order to satisfy the goal, it implements the functionality and technology.

If the work turns out to be harder than the team had expected, then the team might only partially implement the functionality. At the Sprint Review meeting, management, customers, and the Product Owner review how and to what degree the functionality has been implemented. They review how the Sprint Goal has been met. If they are dissatisfied, they can then make decisions about requirements, technology or team composition. During the Sprint, though, the team alone determines how to meet the Sprint Goal. At the end of the Sprint, any incomplete work returns to the Product Backlog.

The team then devises the individual tasks that must be performed to build the product increment.

Activity ID: *7.1 Facilitate Sprint Planning Meeting*

Responsible: *ScrumMaster*

Purpose: A Sprint planning meeting initiates every Sprint. The ScrumMaster is responsible for setting up this meeting, ensuring its successful operation, and concluding the meeting with the Scrum team underway on an agreed to next Sprint. The ScrumMaster ensures that all rules are adhered to and the correct artifacts are created.

Assumptions: The ScrumMaster is familiar with the rules, practices, and process of Scrum.

Risks: The Product Owner may be unavailable for the meeting. Do not conduct the meeting without the Product Owner or an effective delagee of the Product Owner who has full authority to act in their absence. It is better to let the team languish after the end of the Sprint rather than start them on some work that is not important enough for the business to prioritize and describe.

Description: The Sprint planning meeting is crisp and brief, composed of two equal parts, and not lasting more than one day. The Product Owner (or responsible delegate), ScrumMaster, and development team must attend. Others can attend and participate, but their attendance is not required. The meeting has the following parts and components:

1. Define next Sprint - first part of meeting, usually lasting 2-4 hours.

    - The Product Owner presents the highest priority backlog

    - The team asks questions until they understand what was presented in enough detail to understand how to build it

    - The team suggests alternate backlog that may better fit into the Sprint. Such as, “if we do this also, it will be almost free since we will have that code open anyway.”

    - The team and Product owner define an objective for the Sprint, called a Sprint Goal, that is to be met when the functionality is built. The Sprint Goal provides wiggle room for the team to remove or add functionality as time permits

    - The team and Product Owner finalize what product backlog is included within the Sprint and falls within the Sprint Goal.

2. Define the work necessary to turn the product backlog into an increment of potentially shippable product functionality.

    - The team defines what work has to be completed to meet the Sprint Goal. The work consists of tasks, each task requiring one to sixteen hours to complete. If a task takes more than sixteen hours, break it down into multiple tasks or wait until the details are better known so that it can be subdivided at that time.

- define architectures and designs to bond the work together or to further understand the detailed work that must be completed to deliver the functionality

- team members sign up for tasks based on their interests, skills, and input needed to accomplish the Sprint Goal

- the team members estimate the work for all of the tasks and ensure that all dependencies have been thought through; that is, if something has to happen for a task to start, that there is a task for that work.

- the team asks the Product Owner for clarification, as needed, and collaborates with the Product Owner to determine that the work seems right for the goal.

- the team enters the work as Sprint Backlog into a Sprint Backlog spreadsheet that will be used to track work burndown during the Sprint.

3. Once all of this has been accomplished, the Sprint is declared to be started. Before everyone disperses, you establish and publish a date for the Sprint Review meeting and the next Sprint Planning meeting.

As ScrumMaster, your job is to ensure that a meeting that follows this agenda is held to start every Sprint. Guidelines you should enforce are:

1. The primary dialog is between the Product Owner and team. Others may participate and elucidate, but their participation should not diminish the dialog and collaboration between the Product Owner and team. If necessary, limit their involvement by classifying them as “chickens.”

2. Prior to the meeting you should meet with the Product Owner and ensure that the Product Backlog is up to date and ready to present to the team.

3. Ensure that the three artifacts of selected Product Backlog, Sprint Goal, and Sprint Backlog are completely in place prior to ending the meeting.

4. Ensure that the teams has thought through all aspects of turning the selected Product Backlog into potentially shippable product functionality and assigned tasks to them. Include environmental and engineering practice development tasks as well as any training that is needed.

5. Ensure that this is an active collaboration between two partners. The conclusion is an agreement documented in the artifacts.

6. Move the meeting along so that it concludes in less than one day. Don’t let the meeting get bogged down by analysis and design work that will occur within the Sprint.

7. Remember that the team will do its best to meet the Sprint Goal, but it is the team working to make their commitment. If the team can’t make the commitment by the end of the Sprint, this is a learning experience that will help the next Sprint be planned better. It is not a failure or something to be punished.

Activity ID: *7.2 Present Product Backlog*

Responsible: *Product Owner*

Purpose: The Product Owner communicates to the team what requirements are desired in the next product increment delivered at the end of the upcoming Sprint. These requirements are listed in the Product Backlog. The Product Owner presents these items and describes them in enough detail for the team to understand what is desired and to begin work.

Assumptions: The Product Owner has the authority to communicate top priority product backlog and initiate the Sprint with it. The Product Owner has worked with the team to construct reasonable estimates for the top priority product backlog prior to the meeting.

Risks: The team lacks adequate business domain knowledge to collaborate in any meaningful way with the product owner. If this is the case, the team must be reconstituted or augmented to include members with such domain knowledge. Another risk is that the Product Owner has not worked with the team to develop meaningful estimates for the Product Backlog. In this case, the amount of backlog that the team selects may vary significantly from the expectations of the Product Owner.

Description: To start the meeting, the Product Owner presents the top priority Product Backlog. The amount of Product Backlog presented is approximately equal to the sum of estimates for those backlog items compared to the number of working days multiplied by the size of the team. The latter value is a crude estimate that presents a starting point for collaboration.

The Product Owner leads a discussion about what changes to the backlog are appropriate, given what was demonstrated at the end of the previous Sprint (see *Sprint Review*). What does everyone want the team to work on next? What opportunities does the previously completely Sprint offer to the Backlog. The Product Owner works with the team to identify and explain backlog that the team believes it can develop during the next Sprint (30 calendar days).

The Product Backlog is a list of requirements that includes functional and nonfunctional items. Ensure that the nonfunctional items that will scope and bound the functionality are prioritized as high as the functionality that they will support. For instance, if sub-second response time for a large user base is required for a function, these both are the same priority.

Activity ID: *7.3 Select Product Backlog for Sprint*

Responsible: *Team*

Purpose: The team is responsible for selecting as much Product Backlog as it believes and estimates it can turn into an increment of potentially shippable product code during a thirty day Sprint. The Product Owner has presented the top priority product backlog. The team questions the Product Owner to ensure it adequately understands the backlog so that it can make a reasonable commitment. Then the team selects those Product Backlog items.

Assumptions: The team has the business and technical domain knowledge to make estimates for building the various product backlog items presented. If the team does not have such domain knowledge, then additional nonfunctional product backlog items should be established as top priority that address this shortcoming - through training, working sessions, additional analysis, or whatever techniques are deemed appropriate *by the team*.

Risks: The team feels compelled to accept the estimates on the Product Backlog as is, especially if they helped develop them previously with the Product Owner. Nothing focuses attention like a noose, and making a commitment will cause the team to inspect its previous estimates with renewed vigor and accuracy. The team must never commit unless it feels it has a reasonable chance, as a team, to build the functionality during the upcoming Sprint.

Description: The team listens as the Product Owner presents what he or she believes the team can develop in the next Sprint. Ask questions and clarify what is being presented until you have a sense of how to convert these requirements into working functionality. Document these further details for your use during the Sprint.

As the Product Backlog is presented, you may see opportunities to add lower priority product backlog to the Sprint at a relatively modest cost, since you will be working in that area of the system and code anyway. Suggest this to the Product Owner.

As you question the Product Owner regarding the Product Backlog, revise the estimates if necessary. You have just completed a Sprint and may have more insight into the details behind the estimates.

When you estimate how much Product Backlog you should commit to for the Sprint, take into account all of the following:

1. Any vacation, training, corporate meetings, or other down time to which you have already committed during the Sprint;

2. The skills and competence of the various team members;

3. The amount of teamwork that has occurred in the past;

4. The degree to which every and all of the team members can commit themselves to the work. For instance, if a team member has a very sick relative, they will be naturally distracted during the upcoming Sprint.

Based on all of these factors, assess your capacity for work during the Sprint and take that into account when evaluating the estimated work to which you are committing yourself.

Activity ID: *7.4 Define the Sprint Goal*

Responsible: *Product Owner*

Purpose: The Sprint Goal describes the umbrella objective of the Sprint. The Product Owner has proposed Product Backlog and through collaboration with the team has agreed on what will be built during the upcoming Sprint. Why is this backlog being automated, though. What objective will it accomplish, what purpose will be achieved. This is the Sprint Goal, a simple sentence or phrase that describes the purpose of the Sprint. Within this goal, the team will implement more or less of the selected functionality to achieve the goal.

Assumptions: The selected Product Backlog is cohesive and its development will accomplish a goal. If this Product Backlog isn't cohesive, the Sprint Goal may have to be stated in very generalized terms, such as "Implement various functionality that will improve the ease of use of the system."

Risks:

Description: Having selected the Product Backlog, a Sprint Goal is crafted. The Sprint Goal is an objective that will be met through the implementation of the Product Backlog. For example, if the selected Product Backlog is:

To meet the above Sprint Goal, some of the Sprint Backlog that the team might devise is:

- Map the transaction elements to backend database tables;
- Write a business object in C++ to handle transactions via defined methods and interfaces.
- Wrap the C++ in a CORBA wrapper;
- Use Tuxedo for all queueing, messaging, and transaction management; and,
- Measure the transaction performance to determine whether scalability requirements could be met.

Then, an appropriate Sprint Goal could be:

Sprint Goal: to provide a standardized middleware mechanism for the identified customer service transactions to access backend databases.

The reason for having a Sprint Goal is to give the team some wiggle room regarding the functionality. For example, the goal for the above Sprint could also be: "Automate the client account modification functionality through a secure, recoverable transaction middleware capability." As the team works, it keeps this goal in mind. In order to satisfy the goal, it implements the functionality and technology. If the work turns out to be harder than the team had expected, then the team might only partially implement the functionality. At the Sprint Review meeting, management, customers, and

the Product Owner review how and to what degree the functionality has been implemented. They review how the Sprint Goal has been met. If they are dissatisfied, they can then make decisions about requirements, technology or team composition. During the Sprint, though, the team alone determines how to meet the Sprint Goal. At the end of the Sprint, any incomplete work returns to the Product Backlog.

Activity ID: *7.5 Construct Sprint Backlog*

Responsible: *Team*

Purpose: The team and the Product Owner have agreed on what will be developed during the upcoming Sprint. The team now figures out how it will build this functionality, what specific tasks and units of work must be completed for the functionality to live.

Assumptions: The team has the skills and expertise necessary to figure out the work. If not, reformulate the team to include such skills.

Risks: The team may discover that the aggregate of tasks it defines represents more work than available time in the Sprint. If this is the case, the team should review its work breakdown with the ScrumMaster and Product Owner to validate their understandings and assumptions. If there is still too much work, the team and Product Owner should collaborate to reduce the work to an acceptable size for that Sprint. If there isn't enough work, the team should collaborate to increase the backlog.

Description: After establishing the Sprint goal, the team determines what work will have to be performed in order to reach the goal. All team members are required to be present when this is determined. The team may also invite other people to attend in order to provide technical or domain advice. The Product Owner often attends, too, but this is the team's meeting. It is often in this meeting that a team realizes that it will either sink or swim as a team, not individually. The team realizes that it must rely on its own ingenuity, creativity, cooperation, collaboration, and effort. As it realizes this, it starts to take on the characteristics and behavior of a real team. During this meeting, management and the user should not do or say anything that takes the team off the hook.

The team compiles a list of tasks it has to complete to meet the Sprint goal. These tasks are the detailed pieces of work needed to convert the Product Backlog into working software. Tasks should have enough detail so that each task takes roughly four to sixteen hours to finish. This task list is called the Sprint Backlog. The team self-organizes to assign and undertake the work in the Sprint Backlog. Sometimes only a partial Sprint Backlog can be created. The team may have to define an initial architecture or create designs before can fully delineate the rest of the tasks. In such a case, the team should define the initial investigation, design, and architecture work in as much detail as possible, and leave reminders for work that will probably have to be done once the investigation or design has been completed. At that time, the work will be more fully understood and can be listed in more detail.

The team modifies its Sprint Backlog throughout the Sprint. As it gets into individual tasks, it may find out that more or fewer tasks are needed, or that a given task will take more or less time than had

been expected. As new work becomes required, the team adds it. As tasks are worked on or completed, the hours of estimated remaining work for each task are updated. When tasks are deemed unnecessary, they are removed. Only the team can change its Sprint Backlog during a Sprint. Only the team can change the contents or the estimates. The Sprint Backlog is a highly visible, real time picture of the work that the team plans to accomplish during the Sprint.

Sometimes the Scrum Team discovers that it has selected too much Product Backlog to complete in a single Sprint. If this happens, the ScrumMaster immediately meets with the Product Owner. They jointly identify Product Backlog that can be removed while still meeting the Sprint Goal.

Teams become better at Sprint planning after the third or fourth Sprint. At first, a team tends to be nervous about taking on responsibility and it under-commits. As it becomes more familiar with Scrum processes, as it start to understand the functionality and technology, and as it gel into a team, it commits to more work.

Phase Name: *8 Product Backlog Development*

The project starts with a vision of how the product or system will benefit the organization. The best way to realize this vision changes as the project progresses. The business climate may change. Government regulations may introduce new requirements. New technology may change the manner in which the functionality will be developed or operated. As the project progresses, Sprint by Sprint, these changes are reflected in the contents and prioritization of the Product Backlog.

The Product Backlog lists the product requirements. These requirements, or backlog items, are prioritized. The higher priority items will provide more value to the business when implemented, or are necessary prerequisites for this highest value functionality.

The Product Owner is responsible for the Product Backlog. He or she may choose to be the only person who can enter backlog items, or can make the Product Backlog open so that everyone can enter items. Regardless, the prioritization of the Product Backlog is the sole responsibility of the Product Owner.

Activity ID: *8.1 Manage Product Backlog*

Responsible: *Product Owner*

Purpose: Manage the contents and priority of the product backlog to drive each Sprint to deliver the highest priority and most valuable product or system functionality.

Assumptions: The Product Backlog is the only source of work for the development teams. The Product Backlog is stored openly so that its contents are continuously available to anyone interested in the project. Anyone can request that requirements that they have conceived be placed on the Product Backlog.

Risks: If the Product Backlog is not managed, the functionality developed by the teams may not be of the greatest value to the organization.

Description: The requirements that define the most valuable functionality of the system or product change as the environment within which the project occurs changes. The Product Owner assesses these conditions throughout the project, modifies the contents of the Product Backlog accordingly, and prioritizes the Product Backlog to optimize the sequence in which the functionality is constructed and implemented. The following activities are performed by the Product Owner to do so:

Assess Business Conditions -

When the project was conceived and funded, the return on investment of the functionality was measured within the context of known and estimated business conditions. As these conditions change, the value of the functionality that can be developed by teams during upcoming Sprints changes. Several examples are:

- If a competitor releases new functionality that appeals to your organization's customers, the Product Backlog may need to be adjusted to develop and release similar or better functionality sooner than planned, or at all if the functionality hadn't been planned;

- If interest rates soar, the cost of funds may exceed the value of the functionality to be implemented and only some of the functionality may be worth developing;

- If your organization is acquired by another organization, the whole project may be in question and the Product Backlog may need to be frozen; and,

- When the customers and users see the actual functionality at a Sprint review meeting, they may reassess the value of the functionality in their operations. The Product Backlog may need to be adjusted to take these new understandings into account.

Activity ID: *8.1 Manage Product Backlog*

Assess Technology -

The value of some functionality is derived from the technology on which it is implemented. For instance, the primary benefit of defense, avionics, and telephone functionality may be that it is delivered using wireless GPS technology. However, the standards under which this functionality and technology will be used may change as the project progresses, requiring the technology to be reimplemented or adjusted.

If the value of the project diminishes unless the technology is changed, or if using newer technology may provide a greater value than the initially utilized technology, the Product Owner may need to revisit the Product Backlog. Additional Product Backlog that consists of nonfunctional requirements to utilize the new technology may need to be inserted and prioritized.

Assess Additional Requirements Requests -

As the project progresses and becomes increasingly visible through Sprint Reviews and early implementations, many people with interests in the project will think of new requirements for the system or product. Some of these may be minor, such as changing the layout of a user interface. Others may be major, such as the redesign of credit checking. Regardless, all of these requirements need to be captured and entered onto the Product Backlog. This ensures that the Product Backlog represents the needs and thinking of all involved and interested parties, and removes the Product Owner from having to decide which requirements to include and which to exclude. All requirements are valuable, just some more valuable at that time than others.

Maximize Return on Investment through Prioritization -

The Product Owner maximizes the value of the work done by the development teams by ensuring that the Product Backlog that they select during the Sprint Planning meeting is of the highest priority and greatest value to the organization at that moment. At the start of every Sprint Planning meeting, the Product Owner is responsible for presenting this highest value Product Backlog first.

The Product Owner should keep Product Backlog priorities up-to-date. The Product Backlog is kept visible to everyone and anyone throughout the project. Up-to-date prioritization keeps everyone aware of the value decisions made by the Product Owner and also keeps them aware of the assessed value and timetable for functionality in which they are particularly interested.

Phase Name: *9 Sprinting to Develop Product Functionality*

The team works for a fixed period of time. The team has to determine how to achieve its goals and bears full responsibility for making sure that its goals are achieved.

The Scrum Team has decided what it will accomplish during the upcoming Sprint. It now Sprints to accomplish the Sprint Goal. The team is free to accomplish this goal as it sees fit, adapting to the circumstances, technology, and organizational terrain as best it can.

During conflicts, the military will put teams of soldiers into insertion points in areas of operations. Each team is assigned a mission to accomplish and self-organizes to accomplish it. The team has all the supplies and training that it is expected to need. Since the insertion point is usually in the middle of a complex, even chaotic, situation, the team's knowledge of the situation or what to do to reach the goal is limited to a game plan. The team is intended to improvise in order to accomplish its mission. At some predetermined time, the mission ends and the team is picked up.

Scrum was first described in similar terms: "Typically, the process starts with management giving the project team a broad goal. Rarely do they hand out a clear-cut new product concept or a specific work plan. Thus, while the project team has extreme freedom, it is also faced with extreme challenges embodied within the goal. The project team is typically driven to a state of 'zero information' as the extent of the challenge essentially makes prior knowledge inapplicable. Thus the team must fend for itself and find a way to coalesce into a dynamic group."

"According to several of the companies surveyed, the process tends to produce significant quantities of mistakes. However, these are viewed invariably from the plus side as being valuable learning experiences. In the end, the bottom line is that the chaotic process tends to produce more revolutionary products faster than the old sequential development process. It also tends to develop the project team members into 'triple threat' players as each person's knowledge base is broadly expanded through their interaction. At the same time this heightened knowledge filters into the entire organization. (**The New New Product Development Game**, Hirotaka Takeuchi and Ikujiro Nonaka, *Harvard Business Review* (January 1986), pp. 137-146. (WJA)).

Subject: 9 Sprinting to Develop Product Functionality

Sprint Mechanics

Sprints last for thirty calendar days. A team takes this long to get its arms around a problem and to produce a product increment. Management usually can't refrain from interfering if more than thirty days goes by, so the Sprint is limited to thirty days. First-time Scrum users usually want to change the length of the Sprint to, say, sixty days, two weeks, or one week. It is worth resisting this temptation. Thirty days is an excellent compromise between many competing pressures. Adjustments can be made to the duration after everyone has more experience with Scrum.

Every product development project is constrained by four variables, (1) time available, (2) cost, in people and resources, (3) delivered quality, and (4) delivered functionality. A Sprint greatly restricts the first three variables. The Sprint will always be thirty days long. The cost is pretty well fixed to the salaries of the team members and the development environment. This is usually in place before a Sprint starts. However, teams can add the cost of consultants or tools during Sprints to remove impediments. Quality is usually an organizational standard. If it isn't, the team needs to devise quality targets prior to Sprinting.

The team has the authority to change the functionality of the Sprint so long as it meets its Sprint Goal. The team does this is by decreasing or increasing the depth of the functionality delivered. For example, the team can change the depth of functionality to "check account balance." The team can implement this functionality by checking all possible accounts, or only one account. The design and code to perform each implementation is significantly different. At the Sprint Review meeting, the depth to which the functionality is implemented is demonstrated and discussed. Any remaining, unimplemented functionality is reentered onto the Product Backlog and reprioritized.

During the Sprint, all work that is performed is measured and empirically controlled. More or less work may end up being accomplished depending on how things proceed. Factors influencing the amount of work accomplished include the team's ability to work together, the skills of team members, the details of the work to be performed, and the capability of the tools and standards with which the team has been provided. Because Scrum allows the team to change the amount of work it performs during the Sprint, the team has some flexibility, and is able to do more or less so long as it meets its Sprint Goals.

The team is required to deliver a product increment at the end of the Sprint. Daily product builds are an excellent way for the team to measure its progress. Prior to the build, the team should update the test suite and follow each product build with a smoke, or regression, test. Performing code check-ins for the builds is also a good idea, as it improves team communication and coordination.

Activity ID: *9.1 Develop Increment of Functionality*

Responsible: *Team*

Purpose: The team has a list of functionality that it committed to develop into functionality during the thirty day Sprint. The thirty days has started, the clock is ticking. No one is going to tell the team how to develop and test the functionality. For better or worse, it's up to the team to figure out how. The team constructs a list of tasks that, if completed, will result in the functionality. Now the team works together to make this happen.

Assumptions: The team has the collective competence and intelligence to fulfill its commitment to use the technology to build the functionality. Also, the Product Owner, customers, and others with product and business domain expertise are available to answer questions and make decisions.

Risks: If the team doesn't have access to domain expertise, the team is authorized to make the best decision possible based on its understanding of the domain. Although this may result in improper or less than optimal decisions, Scrum's proclivity for action calls for the team to move forward. Once the Sprint has been planned and authorized by the Product Owner, the ball is in the team's court and they are compelled to act and proceed.

Description: After the overall goals and objectives are established at the Sprint planning meeting, the team is dropped into the Sprint. The team is asked to do its best to turn the complex requirements and unpredictable technology into a product increment. It is asked to tame chaos, to turn complexity into predictable product.

Scrum asks people to try to wrest a predictable product from unpredictable complexity. Some people can't handle this type of assignment. During the Sprint, they may decide that they want out. Other people relish the chance to build something that requires their best effort. Those who succeed at Scrum are the individuals that will form the core of an organization. Scrum helps identify these people.

The team has complete authority during the Sprint. It can work as many hours or as few hours as it wants. It can hold meetings whenever it wants. It can hold design sessions from 6am to 10pm. It can spend days interviewing vendors and consultants, or surfing the web for information. The team has absolute authority, because management has given the team free reign for thirty days.

The team has the authority to change the functionality of the Sprint so long as it meets its Sprint Goal. The team does this is by decreasing or increasing the depth of the functionality delivered. For example, the team can change the depth of functionality to "check account balance." The team can implement this functionality by checking all possible accounts, or only one account. The design and code to perform each implementation is significantly different. At the Sprint Review meeting, the depth to which the functionality is implemented is demonstrated and discussed. Any remaining, unimplemented functionality is reentered onto the Product Backlog and reprioritized.

Activity ID: *9.1 Develop Increment of Functionality*

The team has two mandatory accountabilities during the Sprint: (1) Daily Scrum meetings and (2) the Sprint Backlog. These are working tools for the team. Daily Scrum meetings must be promptly attended by all team members, whether in person or via telephone. Team members cannot just send in a passive status report, such as by email or fax. The Sprint Backlog must be kept up-to-date and as accurate as the team's activities, so that it constitutes an accurate and evolving picture of the team and the work that it is doing.

Management has invested thirty days of a team in the Sprint. Regardless of what the team accomplishes, it has acquired valuable working knowledge of the requirements and technology. Even when the team produces nothing tangible, it has nonetheless gone through a very useful learning process. The team has trained itself to take another crack at a reconstituted Sprint goal. It has a deeper understanding of the terrain and the complexity and is better equipped for future Sprints.

Activity ID: *9.2 Maintain the Sprint Backlog*

Responsible: *Team*

Purpose: The team modifies its Sprint Backlog throughout the Sprint. As it gets into individual tasks, it may find out that more or fewer tasks are needed, or that a given task will take more or less time than had been expected. As new work becomes required, the team adds it. As tasks are worked on or completed, the hours of estimated remaining work for each task are updated. When tasks are deemed unnecessary, they are removed. Only the team can change its Sprint Backlog during a Sprint. Only the team can change the contents or the estimates.

Assumptions: The team has an easily available, easy-to-use tool on which to maintain the Sprint backlog. Excel spreadsheets or index cards often fit the need.

Risks: As the team proceeds, it may discover that it has far more work on its hands than time remaining in the Sprint.

Description: The Sprint backlog is a list of work, or tasks, that the team has defined as necessary to turn the selected product backlog into operational functionality. This list includes all of the analysis, design, testing, coding and documentation necessary to build a potentially shippable increment of product functionality. This means that the tasks must include all work to produce high quality software. Refactoring and testing are an integral part of design and coding, not an add-in that can be dropped under time pressure. If time pressure does occur, and the team has more work than it can accomplish, functionality is dropped or made less deep or wide. The anticipated quality is not compromised.

As a team member works on a task, the team member may realize that other tasks are required. In such cases, these tasks are immediately entered onto the Sprint backlog and estimated. At the end of each day, the team member updates the tasks that he or she has worked on. The hours of work remaining is updated to reflect the team member's new estimate of how much work remains left to complete the task. This may be more than the initial amount of work estimated, even after the day's work. That's fine ... hours remaining is a workload estimate, not a time reporting tool.

The Sprint backlog has the tasks listed on the vertical axis and the days of the Sprint listed on the horizontal axis (see example). All of the tasks derived during the Sprint planning meeting are initially listed with their estimates entered in the first day column. When they are worked on, the estimated work remaining is updated daily. When new tasks are defined, they are also entered on the Sprint backlog. Their estimated work is placed in the day of the Sprint when they were initially defined, and updated thereafter as they are worked on.

Activity ID: *9.2 Maintain the Sprint Backlog*

Sprint1 Backlog

Task Description	Responsible	Status (Not Started /	Hours of work rem									
			1	2	3	4	5	6	7	8	9	10
Title Import			550	636	786	772	772	772	770	712	518	510
Write CD of old code	Allen	Completed		1	1	1	1	1	1	1	1	1
Back up Database onto CD	Barry	Completed	10	10	10	10	10	10	10	0	0	0
Rebuild Database	Barry	Completed	0	0	40	40	40	40	40	40	25	25
Blow away directory structure	Allen	Completed	2	0	0	0	0	0	0	0	0	0
Install cruise control and configure (investigate CVS structure; configure cruise control for CVS environment)	Richard	Completed	2	1	1	4	4	4	4	8	0	0
Install and configure Jboss on E10K	Richard	Completed	2	2	2	2	2	2	2	2	2	2
Establish code standards and review with the team	Tim	Completed	2	0	0	0	0	0	0	0	0	0
Review Junit with the team, and construct a sample Junit test	Jim	Completed		0	0	0	0	0	0	0	0	0
Configure IWS for our environment	Jim	Completed		4	4	4	4	4	4	4	4	4
Explore Siteminder, and explain how we use Siteminder with IWS and Jboss.	George	Completed	1	4	4	4	4	4	4	4	3	3
Explore CE. Explain how we use CE in our environment.	Allen	Completed	4	3	3	2	2	2	2	2	2	2
Build a CE prototype.	Allen	Completed	4	4	4	4	4	4	4	4	4	4
Explore Toplink, Explain how we use Toplink in the Land System.	Richard	Completed	4	4	4	4	4	4	4	4	4	4
Build a Toplink prototype.	Richard	Completed	4	0	0	0	0	0	0	0	0	0
Build database schema	Barry	Completed			40	40	40	40	40	40	35	35
Build function that allows importing of titles from LTO data to object graph	Jim	Completed		24	24	24	24	24	24	24	0	0

Activity ID: *9.3 Assess Sprint Burndown*

Responsible: *ScrumMaster, Team, Product Owner*

Purpose: Once the Sprint backlog is constructed during the Sprint planning meeting, the hours for all tasks can be summed. This is the initial estimated work remaining for the Sprint. This workload should be zero by the end of the Sprint, indicating that all work is completed. By assessing the reduction, or burndown, of work daily, the team can assess whether it is likely to be complete by the end of the Sprint. This "burndown" chart is a graphic representation of what every team member intuits from doing the work.

Assumptions: The team has an easily available, easy-to-use tool on which to maintain the Sprint backlog. Excel spreadsheets are best for an easy to use tool from which trend lines can be created, predicting the future based on past work burndown within the Sprint.

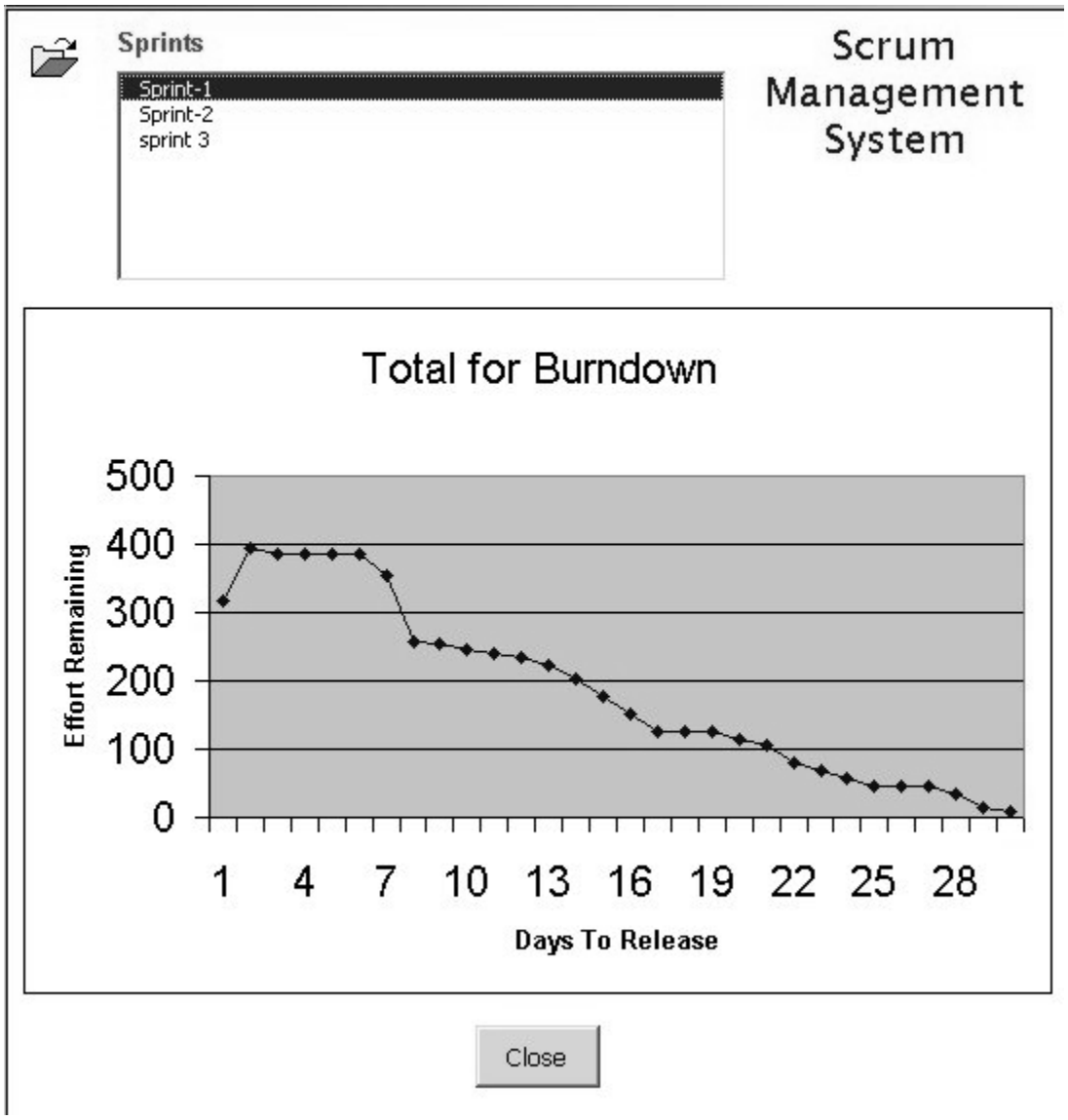
Risks: The work reduction, or burndown, is only effective if each team member updates the hours of work remaining estimate daily as they work on a task. Without daily updating, the burndown is misleading, usually reflecting less progress than actual and leading to managerial distress.

Description: The worksheet below shows a Sprint backlog items. The first column is the Sprint Backlog item. From left to right, the remaining columns are used to contain the amount of work remaining during the Sprint, from day one to day 30. Some of them are being worked on and the number of hours estimated to remain is changing. Others haven't started yet. The total hours estimated as remaining is summed in the bottom row:

Construct an Upload JSP to upload LTCS data files.	4	4	4	4	4	4	4	4	4	4
Build a web page; determine what information to	6	10	10	10	10	10	10	10	10	10
Product backlog for next 12 months	6	6	6	6	6	6	6	6	6	4
Prioritize backlog	6	6	6	6	6	6	6	6	6	4
Determine releases	8	8	7	7	7	7	7	7	6	6
Technical overview for Scrum - ESB Logical Model - Logical DB - Environment Configuration	0	16	16	16	16	16	16	16	16	16
"Bubble" diagrams - Narratives - Glossary	90	78	64	60	60	60	60	50	10	10
Job shadowing	4	4	4	0	0	0	0	0	0	0
Finish project statement	40	40	40	40	40	40	40	40	40	40
Develop Logical Model for Staging Tables	48	46	40	40	40	40	40	40	40	40
Determine how the Contract Management releases of Dovetail will impact the new Land System	5	5	5	4	4	4	3	0	0	0
Request for JDK 1.3.1 with necessary Solaris	1	0	0	0	0	0	0	0	0	0
A								2	2	2
Total for Burndown	275	318	393	386	386	386	385	355	258	254

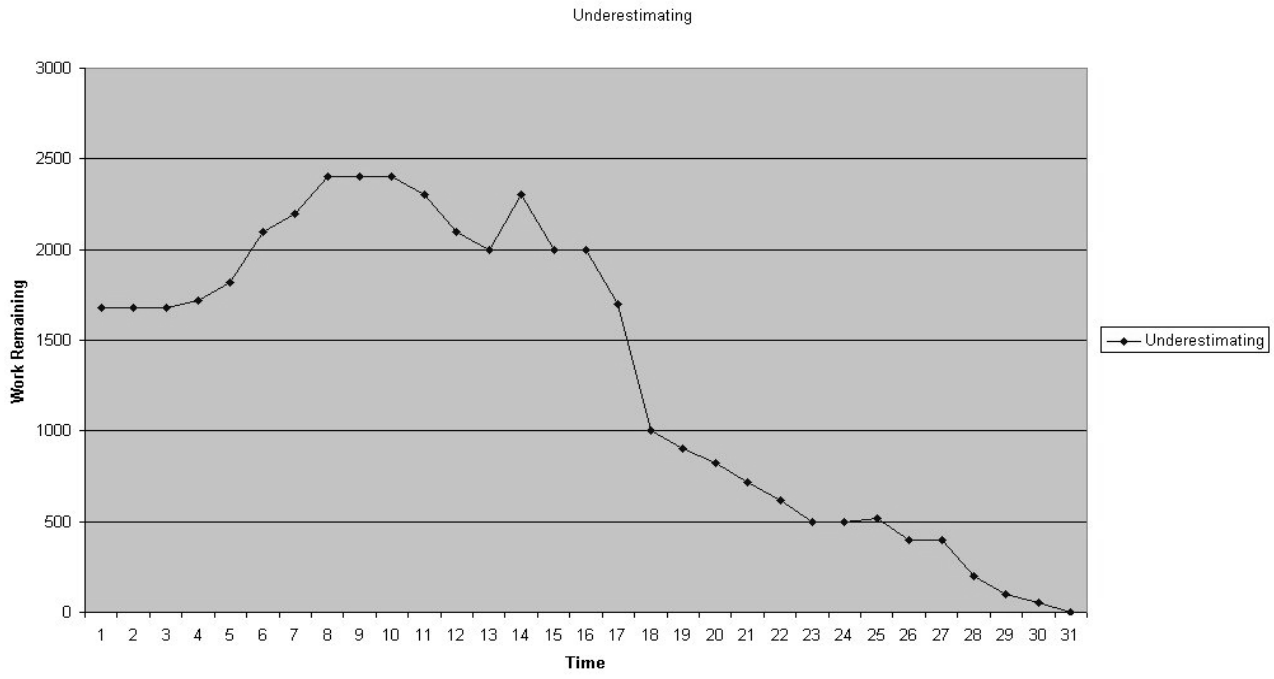
Activity ID: *9.3 Assess Sprint Burndown*

For the Sprint in the previous figure, the burndown graph created by mapping days remaining with estimated work remaining resulting in the following burndown graph:



Activity ID: *9.3 Assess Sprint Burndown*

In some cases, the burndown graph isn't so favorable, indicating that enough progress isn't being made in completing the estimated work, and that at the end of the Sprint all work, and probably all functionality, won't be completed. The following example show such a burndown graph. By the fourteenth day, it is obvious that the team has more work than it will be able to complete (or the team isn't updating the estimated hours remaining faithfully):



.

Activity ID: *9.4 Readjust Commitments*

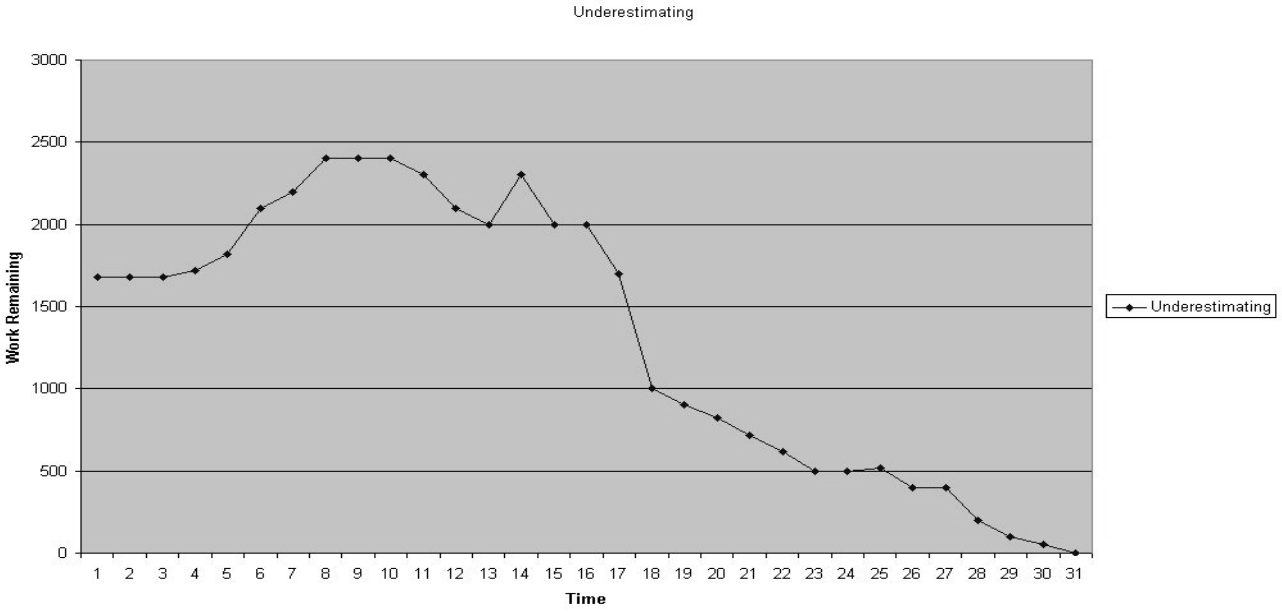
Responsible: *Product Owner, Team*

Purpose: The team has bitten off more than it can chew and turn into product functionality in one Sprint. The team isn't allowed to reduce quality, to increase time or cost, so the only thing left is functionality. The team is required to meet with the Product Owner and assess if work required to implement some or all functionality can be reduced or limited *while still meeting the Sprint Goal.*

Assumptions: The team has been updating the Sprint backlog estimates faithfully and regularly and the estimated work remaining is accurately portrayed.

Risks: Not enough wiggle room was left in the Sprint Goal for adjusting functionality and still meeting the goal. In this case, the ScrumMaster may choose to abnormally terminate the Sprint, call another Sprint planning meeting, and reformulate the Sprint.

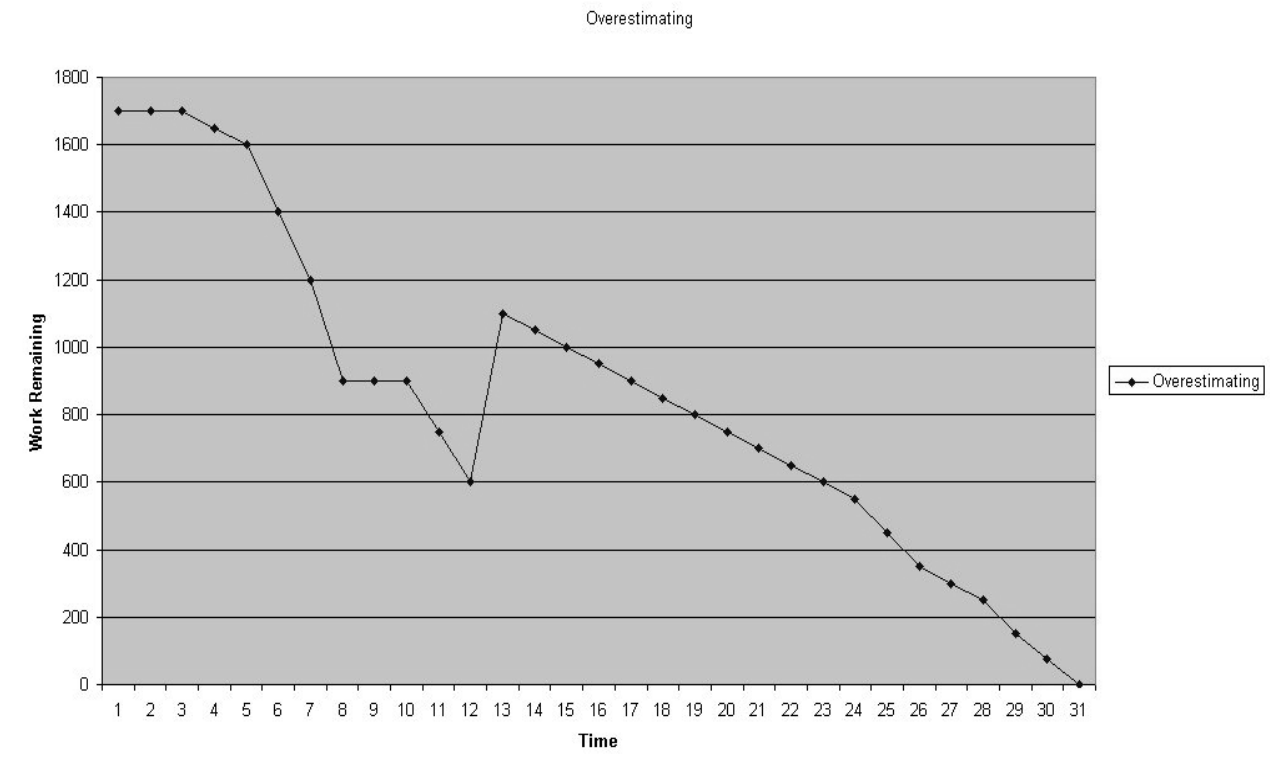
Description: In the below example, the team, ScrumMaster and the Product Owner got together and assessed the situation. The team had committed to automating three transactions on complex, new technology. The team didn't have the domain expertise in the data and found it to be far more complex than anticipated. However, the Sprint Goal was to demonstrate the technology operational for transactions, so the Product Owner was able to reduce the scope of transactions that the team was trying to analyze and automate. Day 16 reflects that the work to automate these



transaction was eliminated.

Activity ID: *9.4 Readjust Commitments*

In the next example below, the team overestimated the amount of work to build the product functionality. All work will be completed prior to the end of the Sprint. Although the Sprint could be terminated early, that would break the heartbeat regularity of deliverables. The team, ScrumMaster and the Product Owner got together and assessed the situation. The Product Owner was able to identify additional Product Backlog that the team committed to automating that was within the "wiggle room" of the Sprint Goal. The Sprint backlog had tasks added to it to address this additional product backlog on the thirteenth day, as reflected in the burndown graph.



Activity ID: *9.5 Abnormal Sprint Termination*

Responsible: *ScrumMaster*

Purpose: Sometimes it is impossible or impractical for the team to achieve the Sprint goal. One reason is that the team cannot build the functionality because of technology instability, inadequate team composition, or complexity of requirements. Another reason is change in business conditions that invalidate the purpose of the Sprint goal. Abnormal termination ends such a Sprint.

Assumptions: Proceeding with the Sprint is of less value than ending, reformulating, and restarting another Sprint. The abnormal termination mechanism is used to prevent undue intervention during a Sprint by outside parties adding additional backlog to the team, or preventing the team from floundering and being unproductive.

Risks: Abnormal termination is the responsibility of the ScrumMaster. This is a gutsy call indicating that something has gone awry and needs to be fixed. In most organizations, halting the production line is not an honored tradition, but the ScrumMaster's job is to ensure the process works and produces valuable functionality.

Description:

Sprints can be cancelled before the allotted thirty days are over. Under what kind of circumstances might a Sprint need to be cancelled? Management may need to cancel a Sprint if the Sprint Goal becomes obsolete. A company as a whole may change direction. Market conditions or technological requirements might change. Management can simply change its mind. In general, a Sprint should be cancelled if it no longer makes sense given the circumstances. However, because of the short duration of Sprints, it rarely makes sense for management to cancel a Sprint.

Sometimes the team itself may decide that a Sprint should be cancelled. A team comes to better understand its abilities and the project's requirements during a Sprint. The team may realize midway through the Sprint that it cannot achieve its Sprint Goal. Even if the team's knowledge of its work has not changed, the Sprint could still need to be cancelled. For example, the team might run into a major roadblock. Sometimes, the team feels that it has met its Sprint goal, and decides to cancel the Sprint because it wants more direction from management before proceeding to implement more functionality.

That the team has the power to ask the ScrumMaster to cancel a Sprint is very important. The team is able to stay focused because it can terminate the Sprint if someone tries to change the nature or scope of its work. Everyone knows this, and is consequently reluctant to make any such changes. Sprint terminations consume resources, since everyone has to regroup in another Sprint planning meeting to start another Sprint. The first question that is asked when a Sprint is terminated is usually "Who is responsible for this meeting occurring early?" Because people don't want to be named as the answer to this question, very few Sprints end up being terminated.

Activity ID: *9.6 Stop External Interference*

Responsible: *ScrumMaster*

Purpose: Once the team commits to the Sprint goal and Product Backlog for the Sprint, it is to be left utterly alone to do its best to fulfill its commitments. The ScrumMaster is responsible for monitoring the team and enforcing this rule.

Assumptions: The primary tool for the ScrumMaster monitoring for outside interference is the Daily Scrum. The ScrumMaster must listen closely during this meeting and assess if outside interference is occurring. If so, the ScrumMaster must redirect the team to continue with its own work. The ScrumMaster must then work with the source of the external interference to ensure that the rules are understood and the interference ceases.

Risks: Most organizations tolerate continual interference in the development process, leading to teams floundering and experiencing reduced productivity. When Scrum is first introduced into an organization, the ScrumMaster is a change agent that must politely but firmly enforce this rule.

Description:

A team is let loose for the thirty day Sprint. The team has committed to the goal and accepted the responsibility of building a product increment. It has the authority to act as it sees fit. No person outside the team can change the scope or nature of the work the team is doing. No one is allowed to add more functionality or technology to the Sprint. No one can tell the team how to proceed in its work.

Many organizations are initially uncomfortable with the idea of letting a team loose for a Sprint. It just doesn't feel right. It feels too risky. Is it really so strange for management to trust a team of its own employees to figure out the best and most appropriate things to do? How much of a risk is this really? Management has assigned the best people available to the team. What the team will do is defined in the Sprint Goals and Product Backlog. The risk is limited to thirty calendar days of the team's Sprint. Management can see how the team's doing by attending Daily Scrums and, failing that, can always inspect the most recently updated version of the Sprint Backlog. At the end of the thirty days, management meets with the team. At the very worse, the team has built nothing. More often, the team has built something that reflects its best efforts. The team often exceeds expectations. Once the creative juices get flowing, teams become hotbeds of creativity and productivity. A Sprint is management's bet that employees are capable and know what they are doing.

Activity ID: *9.6 Stop External Interference*

The team has prepared its Sprint backlog to achieve its goals. At the Daily Scrum meeting, the team reports on its progress on the various tasks in the Sprint backlog. The ScrumMaster should listen closely to ensure that the team members are providing status on its relevant to its commitments and the Sprint Goal. Another source of information is observing the team. If the ScrumMaster detects out-of-commitment work occurring, he or she should meet with the involved team member(s) to 1.) determine the source and nature of the external interference, and, 2.) redirect the team members to continue on the work of the Sprint.

The ScrumMaster is then responsible for meeting with the source(s) of the interference and advising them of the “no interference” rule. Most organizations tolerate interference. Usual sources are upper management or other standard setting professionals, such as database administrators or systems architects. They may, with good intentions, be trying to redirect the team to a better way of doing work, or to include some work that they view as critical to the team.

The team can ask for advice or seek consultation during the Sprint. However, how it does the work to which it committed itself is its responsibility alone. If an outside source is so convinced that the work that the team is doing is wrong or incorrect, the mechanism for them to use is to request that the ScrumMaster abnormally terminate the Sprint.

If, upon meeting with management and the Product Owner, the ScrumMaster determines that an abnormal termination is appropriate, the source is the abnormal termination is made visible at the review meeting that immediately follows the abnormal termination. At that meeting, the source is required to state their case.

Activity ID: *9.7 Remove Impediments*

Responsible: *ScrumMaster*

Purpose: A responsibility of the ScrumMaster is to optimize the team's productivity. Impediments to team productivity that are noticed by the ScrumMaster are to be addressed and removed in an expeditious manner.

Assumptions: Impediments are identified by team members during the Daily Scrum. However, the ScrumMaster is also responsible for actively working to detect additional impediments.

Risks: Scrum often changes the culture of the organization implementing it, from tolerating the status quo to making changes to optimize productivity. The ScrumMaster is a change agent in causing these changes to happen. To the extent that the ScrumMaster does this work professionally and sensitively, the organization benefits. To the extent that the ScrumMaster is unrelenting in helping the team by removing benefits, the team and the organization benefit. Strike a balance.

Description: An impediment is anything that stands in the way of the team or team members being as productive as they can be. Examples are:

1. waiting for software or hardware due to slow purchase order processes;
2. waiting for software or hardware due to slow vendor response;
3. having to attend meetings that weren't taken into account while planning the Sprint;
4. slow response by business decision makers to questions;
5. network being down; and,
6. inadequate working facilities.

The ScrumMaster notes these impediments by observing the team, or by listening to the team members report the impediments during the Daily Scrum. When Scrum is first implemented, both the ScrumMaster and the team may view impediments as business as usual, the normal culture of the organization. The ScrumMaster's job it to make the team and the organization aware of the cost of these impediments, and to help the organization change so the impediments are removed. Impediments should be the exception, rather than the rule.

Phase Name: *10 Daily Scrum*

Each Scrum Team meets daily for a 15-minute status meeting called the Daily Scrum. During the meeting, the team explains what it has accomplished since the last meeting, what it is going to do before the next meeting, and what obstacles are in its way. The Daily Scrum meeting gets people used to team-based, rapid, intense, cooperative, courteous development. Daily Scrums improve communications, eliminate other meetings, identify and remove impediments to development, highlight and promote quick decision-making, and improve everyone's level of project knowledge. That's a lot of benefit from just 15 minutes a day!

The Daily Scrum is the only formal communication between the team and the people outside the team during a Sprint. If anyone wants to assess the progress of the team prior to the end of Sprint Review meeting, they can attend the daily Scrum meeting (as a "chicken") or inspect the Sprint Backlog. Nobody outside of the team is allowed to interfere with the team's time by calling any other type of review meeting, such as a "design review." The ScrumMaster should view such a meeting as an interference and remove the need for any team member to attend.

The Daily Scrum has three purposes:
1. The team members share status with each other.
2. The team members report any impediments or decisions that they can't make to the ScrumMaster so that the ScrumMaster can resolve them.
3. Team members and the ScrumMaster get to assess the team through observation.

By listening carefully during a Daily Scrum meeting, managers can get a sense of what the team is doing and how likely it is to succeed. It is much easier to attend a Daily Scrum than it to read a written report, and Daily Scrums have the additional benefit of being a boon for the team as well as for its managers. Scrum is direct and open. Because the reporting interval is only 24 hours, it's easy to continuously monitor a team and for the team to monitor and coordinate the work of each other. A ScrumMaster can quickly see if a team member is up to his ears trying to get a piece of technology to work or chasing down something for a Vice President. Has a team member lost interest in the project? Is someone not working because of family problems? Is the team quarreling over something? What attitudes are demonstrated during the meeting?

The Daily Scrum was initially thought to be a simple status meeting. However, the sociology of people committing to work daily in front of their peers, and then reporting the next day of their success in performing the work is complex and profound. This simple act helps the team members honestly share their successes and problems, and provides an opportunity for the team members to openly offer to help each other without any negative impression.

The Daily Scrum meeting is brief, usually lasting less than 15 minutes. The ScrumMaster enforces the brevity of the meeting through the manner in which he or she conducts it.

Activity ID: *10.02 Setup Facilities for Daily Scrums*

Responsible: ***ScrumMaster***

Purpose: Ensure an appropriate facility for the Daily Scrum meeting.

Assumptions: The ScrumMaster is innovative and will find such a facility regardless of the obstacles.

Risks: None. If such a facility cannot be established, the organization is not ready for Scrum. Return to the previous manner for conducting development projects.

Description:

The ScrumMaster should establish a meeting place and time for the Daily Scrum. The room in which the Daily Scrum is held is called the Scrum Room. This room is also useful for follow-up meetings so it should be scheduled accordingly, with buffer time after the Daily Scrum.

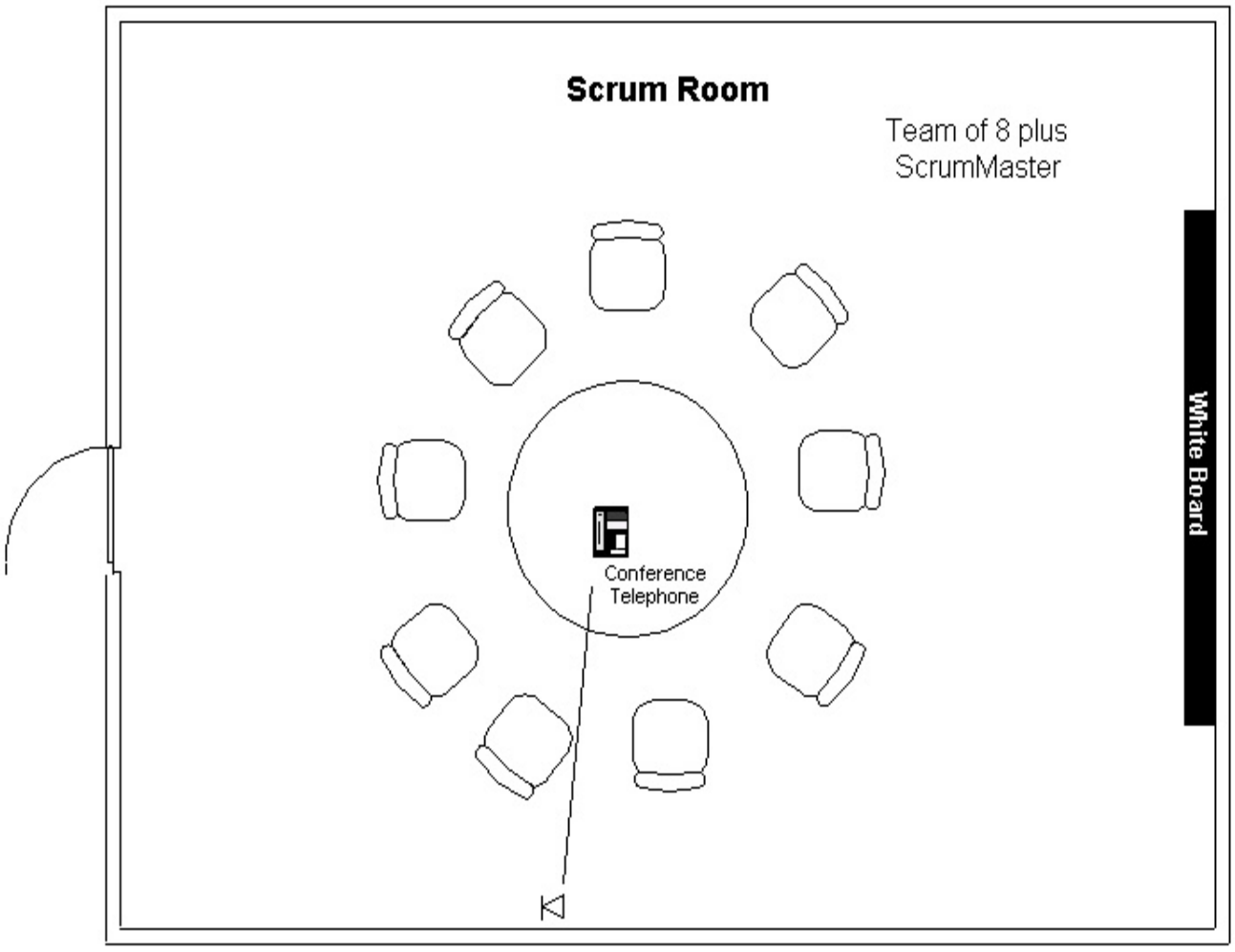
The team will hold its Daily Scrum in this room every working day at the same place and same time. The room should be readily accessible from the team's primary working location. It should be equipped with a door (to close during the meeting), a speakerphone (for team members who will attend by calling-in), a table, at least enough chairs for each team member to sit around the table, and white boards (for recording notes, issues, and impediments and for general brainstorming after the Daily Scrum). Of course, Scrum has been successfully implemented in environments that did not have Scrum Rooms that were this well appointed. The most important thing is that the time and location of the Daily Scrum be constant. If necessary because an optimal facility isn't available, start out by holding Daily Scrums in the corner of a cafeteria, on a lawn, and even in a neighboring coffee shop. It is always only a matter of time before management sees the value of the Daily Scrum and provides a Scrum Room, or if management has already provided one, to improve the facilities in the Scrum Room.

The facility should have adequate space for every team member to attend. Allow additional stand-up space for visitors to attend. The Scrum room can be a dedicated room, or if the team is collocated in open space, the team can meet in an area of the open space.

Activity ID: *10.02 Setup Facilities for Daily Scrums*

Some ScrumMasters prefer that everyone stand during the Daily Scrum, reinforcing the brevity of the meeting. In these circumstances, the ScrumMaster usually removes the chairs.

A Scrum room might look like this for a eight person team:



Activity ID: *10.1 Conduct the Daily Scrum*

Responsible: *ScrumMaster*

Purpose: Conduct a daily meeting at which the team can coordinate its work, exchange status, and request assistance in making decisions or removing impediments. The daily meeting is also an event where people outside the team can observe the team in action.

Assumptions: The ScrumMaster and team have a suitable meeting place where all team members can participate, even those not collocated. A suitable time has been determined for the daily meeting that can be attended by team members who aren't collocated and are in different time zones.

Risks: The ScrumMaster isn't firm enough in conducting the meeting and it becomes a source of external interference to the team, or the meeting takes too long and wastes valuable team time. Everyone hates meetings and the Daily Scrum is a good place to demonstrate what an effective meeting looks like.

Description: The ScrumMaster is responsible for successfully conducting the Daily Scrum. The ScrumMaster keeps the Daily Scrum short by enforcing the rules and making sure that people speak only briefly. This require a fair amount of courage since the rules apply equally to everyone. It's difficult to tell a Senior Vice-President not to interrupt.

The ScrumMaster is responsible for ensuring that the Daily Scrum goes well. Scrum Masters ensure that the room is setup for the meeting. They get any team members working from remote locations set up on a conference phone before the meeting starts[1]. Also, they work to minimize the distractions

Chickens and Pigs

A chicken and a pig are together when the chicken says, "Let's start a restaurant!"

The pig thinks it over and says, "What would we call this restaurant?"

The chicken says, "Ham n' Eggs!"

The pig says, "No thanks. I'd be committed, but you'd only be involved!"

that occur during the meeting so that everyone can stay focused and the meeting can be kept short.

For example, a good ScrumMaster might even set the chairs up around the table before the meeting begins so that people don't get caught up in side conversations as they do move chairs around. The Scrum Master's job is to increase the productivity of the team in any way possible. This is a small demonstration of commitment.

Team members commit to a goal and do the work that is required to meet it. They are called pigs because they, like the pigs in the joke, are committed to the project. Everyone else is a chicken. Chickens can attend Daily Scrums, but they have to stand on the periphery. Chickens are not allowed to interfere with the meeting in any way, such as talking, gesticulating, or making crude noises. Chickens are present as guests and must follow Scrum rules.

The team should arrange themselves in a circle, generally around a focus such as a table. Some Scrum Teams sit, while others have found that standing encourages brevity. Team members seat themselves in any order as they arrive. People not on the team sit or stand around the periphery, outside of the team circle. When guests sit around the table, or interject themselves into the team circle, they feel free to interject comments, to have side conversations. This disrupts the status from the team members and makes it hard to control the meeting duration. If they are placed outside the circle, they are physically reminded that they are observers, and not participants.

Every Pig must be at the meeting on time. The ScrumMaster is responsible for starting the meeting regardless of attendance. Anyone who arrives late or is absent and hasn't given the ScrumMaster their status has to pay $1 to the ScrumMaster. The ScrumMaster keeps these late fees and periodically donates them to charity.

During the Daily Scrum, only one person talks at a time. That person is the one who is reporting his or her status. Everyone else listens. There are no side conversations. Starting to his or her immediate left, the ScrumMaster goes around the room and asks team members to answer three questions.

What have you done since last Scrum? This question addresses only the last 24 hours, unless a weekend or holiday has occurred in the interim. Team members only mention the things that they have done that relate to this team and this Sprint. For example, the team isn't interested in other work that part-timers might be doing unless it relates directly to their own work. If team members are doing work other than what they had planned to be doing for this Sprint, that other work should be identified as an impediment. Anything not related to the team's work is probably an impediment.

What will you do between now and the next Scrum? This question relates only to this Sprint and this team. What is each team member planning to work on? The work that team members expect to do should match the work that has been planned by the team. If team members state that they are going to be doing other work, they should be asked why. The team might need to meet after the Daily Scrum to talk about the new work. Other team members have to adjust their work based on the new work. Getting answers to these questions can helps the team and management assess whether the work is proceeding regularly and as expected, or if adjustments are needed.

What got in your way of doing work? If a team member was unable to work or anticipates being unable to work on what he or she planned, what got in his or her way? That is to say, what is getting

in the way of the team? Each team member has planned and committed to a goal and is empirically figuring out the work to meet the goal. What is slowing down individual team members, and therefore the team as a whole? Although team members have worked within the organization and are used to its culture and style, the ScrumMaster should encourage them to think "outside of the box." If this were the perfect work environment, what else would it have? More specifically, what could help the team be more productive, both as a group of individuals and as a cohesive team?

The ScrumMaster is responsible for keeping Pig responses to these questions brief and to the point. They shouldn't elaborate or describe how the work was done or will be done unless they want to highlight help that they may need. For instance, a team member may report that he or she intends to complete implementing a feature in a module, but he or she is having difficulty understanding how a specific algorithm works. Or the team member may report that he or she is going to check in some code but can't get the source code management system to work without crashing.

The Daily Scrum is not a design session and should not turn into a working session, so don't discuss design or start to solve a problem. There isn't enough time or flexibility in the Daily Scrum to begin working through issues of this magnitude. By limiting the meeting's scope, the ScrumMaster can keep the duration in check and constant. If the scope of the Daily Scrum expands, no one will know how much time to allocate to the meeting.

The rules for the Daily Scrum are:

1. Hold the Daily Scrum in the same place at the same time every work day.

2. All team members are required to attend. If, for some reason, they can't attend in person, the absent member must either attend by telephone or by having another team member report their status for them.

3. Team members must be prompt. The ScrumMaster starts the meeting promptly at the appointed team regardless. Any members who are late pay $1 to the ScrumMaster immediately.

4. The ScrumMaster begins the meeting by starting with the person immediately to his or her left and proceeding counterclockwise around the room until everyone has reported.

5. Each team member should respond to the three questions only (1. What have you done since the last Daily Scrum regarding this project?, 2.) What will you do between now and the next Daily Scrum meeting regarding this project?, and, 3.) What impedes you from performing your work as best as possible?

6. Team members should not digress beyond answering the three questions into issues, designs, discussion of problems, or gossip. The ScrumMaster is responsible for moving the reporting along briskly, from person to person.

7. During the Daily Scrum, only one person talks at a time. That person is the one who is reporting his or her status. Everyone else listens. There are no side conversations.

8. When a team member reports something that is of interest to other team members or needs the assistance of other team members, any team member can immediately arrange for all interested parties to meet after the Daily Scrum to setup the meeting or meet right then.

9. Chickens are not allowed to talk, make observations, make faces, or otherwise make their presence in the Daily Scrum meeting obtrusive.

10. Pigs or Chickens who cannot or will not conform to the above rules may be excluded from the meeting (chickens) or removed from the team (pigs).

The ScrumMaster is responsible for keeping a list of all open impediments and decisions in the Scrum Room, along with when it was first reported, by whom, and status.

Activity ID: *10.2 Commit and Status*

Responsible: ***Team***

Purpose: Provide a daily report on daily status and impediments so as to synchronize and coordinate team activities and optimize productivity.

Assumptions: A Scrum Room is available and all team members promptly attend the meeting and answer the three Daily Scrum questions.

Risks: Learning and respecting the mechanics of an effective meeting, such as the Daily Scrum, requires a shift in attitude and culture in some organizations. Some organizations have not been able to use Scrum because the culture gets in the way of people showing up on time.

Description: Every team member must arrive on time for each Daily Scrum. The meeting starts promptly at the designated time, regardless of who is or is not present. Prior to attending, prepare answers to the three Daily Scrum questions, listed below:

What have you done since last Scrum? This question addresses only the last 24 hours, unless a weekend or holiday has occurred in the interim. Team members only mention the things that they have done that relate to this team and this Sprint. For example, the team isn't interested in other work that part-timers might be doing unless it relates directly to their own work. If team members are doing work other than what they had planned to be doing for this Sprint, that other work should be identified as an impediment. Anything not related to the team's work is probably an impediment.

What will you do between now and the next Scrum? This question relates only to this Sprint and this team. What is each team member planning to work on? The work that team members expect to do should match the work that has been planned by the team. If team members state that they are going to be doing other work, they should be asked why. The team might need to meet after the Daily Scrum to talk about the new work. Other team members have to adjust their work based on the new work. Getting answers to these questions can helps the team and management assess whether the work is proceeding regularly and as expected, or if adjustments are needed.

What got in your way of doing work? If a team member was unable to work or anticipates being unable to work on what he or she planned, what got in his or her way? That is to say, what is getting in the way of the team? Each team member has planned and committed to a goal and is empirically

figuring out the work to meet the goal. What is slowing down individual team members, and therefore the team as a whole? Although team members have worked within the organization and are used to its culture and style, the ScrumMaster should encourage them to think "outside of the box." If this were the perfect work environment, what else would it have? More specifically, what could help the team be more productive, both as a group of individuals and as a cohesive team?

Team members should keep their responses to these questions brief and to the point. They shouldn't elaborate or describe how the work was done or will be done unless they want to highlight help that they may need. For instance, a team member may report that he or she intends to complete implementing a feature in a module, but he or she is having difficulty understanding how a specific algorithm works. Or the team member may report that he or she is going to check in some code but can't get the source code management system to work without crashing.

If a team member cannot physically attend the Daily Scrum, they should either call in to the conference phone in the Scrum Room just prior to the meeting, or - as a last alternative - provide their status to another team member and ask the other team member to report for them.

Establishing Follow-Up Meetings

If any discussion is needed other than the status provided by answering the three questions, a follow-up meeting may be requested. After a team member gives his or her status, another team member can interject, "I'd like to address this more after the Scrum. Anyone who's interested should hang around afterward." More than one team member may want to address the topic in more depth. This conversation may get into discussing design or requirements alternatives or interpretations. A team member may be working on the same thing and wants to share information. This sharing will be of indeterminate length and may lead to design discussions. A team member may have done something like this before or may know an easier way to do the work. This may lead to another team member suggesting another approach, resulting in a design discussion. A working meeting is needed to reach a decision or to discuss design or standards. In each of these cases, the conversation that starts is open-ended. Other team members may join in and more time will go by. All of these discussions are worthwhile and should happen. They should happen after the Daily Scrum, though. Keep all working sessions outside of the Daily Scrum, or else the distinction between a status session and a working session will become blurred and the time for the Scrum won't remain short and fixed.

Activity ID: *10.3 Make Decisions*

Responsible: ***Product Owner, ScrumMaster, Team***

Purpose: Provide a mechanism for team members getting quick resolution to decisions that they are unable to make themselves and cannot otherwise find the appropriate party to make the decision for them.

Assumptions: The ScrumMaster has encouraged the team to make decisions on its own when a quick answer isn't available. Only if the team member is uncomfortable with their domain knowledge and can't get help should decisions be brought to the Daily Scrum. Scrum is intended to encourage the team to do the required analysis and design behind decisions.

Risks: Team members shouldn't feel obligated to make decisions that are holding them up, nor should they sway too far to asking someone else to make all decisions for them.

Description: A Scrum team has the full authority to make all of the decisions necessary to turn the Product Backlog into a Product Increment to meet the Sprint Goal. The team is free to do whatever is necessary to make the best decision possible. The team members can interview others, bring in consultants, read books, browse the web, whatever they need. A team member may identify indecision as an impediment (e.g. "I don't know if I should do this or that."). The ScrumMaster is then responsible for making a decision, preferably then and there. When first implementing Scrum for a team, the ScrumMaster should be careful not to make too many decisions for the team. Delegated decision-making is new in most organizations. The ScrumMaster helps the team learn to make its own decisions to fulfill its commitments. The more the team relies on outsiders to make its decisions, the less control it has over its commitments.

When the team is uncertain, it should acquire whatever information is necessary to become more certain. Sometimes a team asks for a decision to be made when it feels that the decision is risky, or sensitive. In this case, the ScrumMaster should meet with the team after the Daily Scrum and work through to a decision. A team should make a decision by acting on the best information that it has and by relying on its instincts. Most snap decisions are more acceptable than holding up work to wait for someone else to decide. The team usually has a far better handle on the alternatives than anyone else. Also, completed work has momentum and usually will be "good enough," or at the very least, it will be far better nothing.

Most of the time the decision will be acceptable. Sometimes, though, a decision results in unaccept-

able product functionality or application of technology. This becomes apparent when the product increment is reviewed at the end of the Sprint. If an incorrect decision isn't visible at this review, it is probably irrelevant. Otherwise work can be redone to correct the bad decision. Because Sprints are so short, bad decisions rarely impact more than thirty days worth of work.

If the ScrumMaster can't make a decision during the meeting, he or she is responsible for making a decision and communicating it to the whole team within one hour after the end of the Scrum.

Activity ID: *10.4 Remove Impediments*

Responsible: ***ScrumMaster, Team***

Purpose: To quickly identify impediments to the team's productivity and progress.

Assumptions: Team members understand that impediments are not personal weaknesses and that the team should want them removed by management. This is the role of the ScrumMaster and the team should utilize it.

Risks: Initially, team members may accept impediments as part of doing business at their organization. Only when they realize that the ScrumMaster's job is to remove impediments will they start looking for an identifying them. The ScrumMaster should encourage this by observing the team and identifying impediments initially for it.

Description:

If a team member identifies something that is stopping him or her from working effectively, the ScrumMaster is responsible for recording and removing that obstacle. Impediments should be written down on the white board on the wall. If the ScrumMaster doesn't fully understand the impediment, he or she should meet with whoever mentioned it after the Scrum Meeting to learn about it. The following are common impediments:

- workstation, network, and/or server are down;
- network or server are slow;
- required to attend human resource training session;
- required to attend status meeting with management;
- asked by management to do something else;
- asked to do something other than what this team member committed to for this sprint;
- unsure about how to proceed;
- unsure of design decision; and,
- unsure how to use technology.

The Scrum Master's top priority is removing impediments. If team members inform the ScrumMaster that he or she can do something to make them more productive, the ScrumMaster should do it. Every day, the Scrum gives the ScrumMaster direct information on what he or she can do to improve the productivity of the team.

Activity ID: *10.4 Impediments*

If the impediments aren't promptly resolved, the team will report the next day that it is still impeded. It is a bad sign if the team members stop reporting impediments even though they haven't been resolved. This usually means that the team members have lost their confidence that the ScrumMaster can and will resolve their impediments. If, for some good reason, an impediment cannot be removed, the ScrumMaster should report on this at the next Daily Scrum.

If the open impediments on the white board start getting to be a lengthy list, this may indicate that the larger organization isn't supporting the team. In this case, the ScrumMaster may end up having to abnormally terminate the Sprint. This is a very powerful card to play. It should be played only when the ScrumMaster is very concerned that the organization's support for the project is so low as to render the team ineffective and demoralized. Low support could be because this is an unimportant project or because the organization is unable to effectively support any projects. The reason doesn't matter to this project. The ScrumMaster has observed that there are many impediments and management is unwilling or unable to remove them. The ScrumMaster should very carefully and intensely discuss these observations and the consequences of the lack of support with management before canceling the Sprint. Once the decision has been made to cancel the Sprint, the ScrumMaster is effectively stating that the organization shouldn't implement Scrum on this project at this time.

Activity ID: *10.5 Attend the Daily Scrum*

Responsible: *Chickens*

Purpose: People who are not part of the team but who want to be aware of the project status during a Sprint are invited to attend any Daily Scrum meeting. These people are called “chickens” to differentiate them from the “pigs” on the team.

Assumptions: The ScrumMaster has provided a mechanism for anyone interested to learn the location and time of the Daily Scrum meeting for each project team.

Risks: The chickens will be unable to resist talking, helping, and giving direction and guidance as they listen to the status at the Daily Scrum.

Description: Anyone who wants to learn about the status of a project team during a Sprint is invited to attend the team’s Daily Scrum meeting. Anyone attending as a chicken is required to adhere to the following rules, enforced by the ScrumMaster:

1. Chickens are not allowed to make their presence known by any means, such as talking or facial expressions;
2. Chickens stand on the periphery of the team so as not to interfere with the meeting;
3. If too many chickens attend the meeting, the ScrumMaster may limit attendance so the meeting can remain orderly and focused; and,
4. Chickens are not allowed to talk with team members after the meeting for clarification or to provide advice or instructions.

During the Sprint, the Scrum team is responsible for management of its own activities to meet the commitments it made at the start of the Sprint. Except for the case of an Abnormal Termination of the Sprint, nobody has authority over the team during the Sprint.

Activity ID: *10.6 Scrum of Scrums*

Responsible: *ScrumMaster, Team*

Purpose: The Scrum of Scrum meetings coordinate multiple teams in projects that have been scaled above a single team. Just as teams are optimized around seven people, each level of scaling should be held to around seven groups that are coordinating themselves. As the coordinating meetings go up the hierarchy, the frequency of the meetings can decrease from daily to less frequently. Finely granulated coordination happens at the team level, coarsely granulated coordination happens after several levels of elevation.

Assumptions: The project can't be done with a single team, or the number of people who need to be involved exceeds the limits of one team's size.

Risks: Someone must be responsible for coordinating the entire mechanism and ensuring that each level meets frequently enough to synchronize the entire hierarchical mechanism.

Description: Any system can be built with one Scrum team, consisting of approximately seven people. When there are numerous requirements and the time one team would need to develop them is unacceptable, additional teams are formed. People trained in traditional project management practices lay out a PERT chart that defines all the work. Elements that are as orthogonal (Orthogonal, mutually perpendicular) to each other as possible are then identified. Each element will yield code and functionality that's as cohesive as possible and will minimize couplings – or dependencies – on other parts of the system. These orthogonal[1] areas of work form the basis for team assignments.

Scrum uses the same project management practices to allow multiple teams to undertake development. However, the work is apportioned from a systems architecture rather than from a PERT chart. Similar to staging complex system development, construction of systems architecture is staged in the initial Product Backlog. A single team builds the architecture in the initial Sprint and the Product Backlog requirements are mapped to the various orthogonal subsystems. As new requirements emerge, they also are mapped to the subsystems.

Activity ID: *10.6 Scrum of Scrums*

After the first Sprint, management can establish multiple teams. Each team is assigned to one or more orthogonal subsystems. Prior to each Sprint, each team selects work from the Product Backlog, in order of priority, by the subsystem to which they're assigned. This way, teams can develop functionality with as little dependencies and impediments as possible.

Orthogonality is a great concept, but it's rarely achieved in systems development. Things wind up being somewhat dependent on each other no matter how hard team members teams try. Teams must be coordinated and aware of other teams' work, progress, and issues.

Scrum projects have daily status meetings, or "daily Scrums." These are short status meetings of no more than 15 minutes, in which each team member answers three questions: 1) What have I done since the last daily Scrum?; 2) What am I planning on doing between now and the next daily Scrum?, and 3) What is impeding my work? These meetings are used to coordinate work within and across teams.

To coordinate the work between teams, I usually institute "Scrum of Scrum" meetings. After the daily Scrums, another daily Scrum is conducted with a representative from each team. This meeting synchronizes and coordinates the work of various teams. Representatives from each team answer the same three questions, but from their teams' perspectives. What comes out of this meeting is the status of each teams.

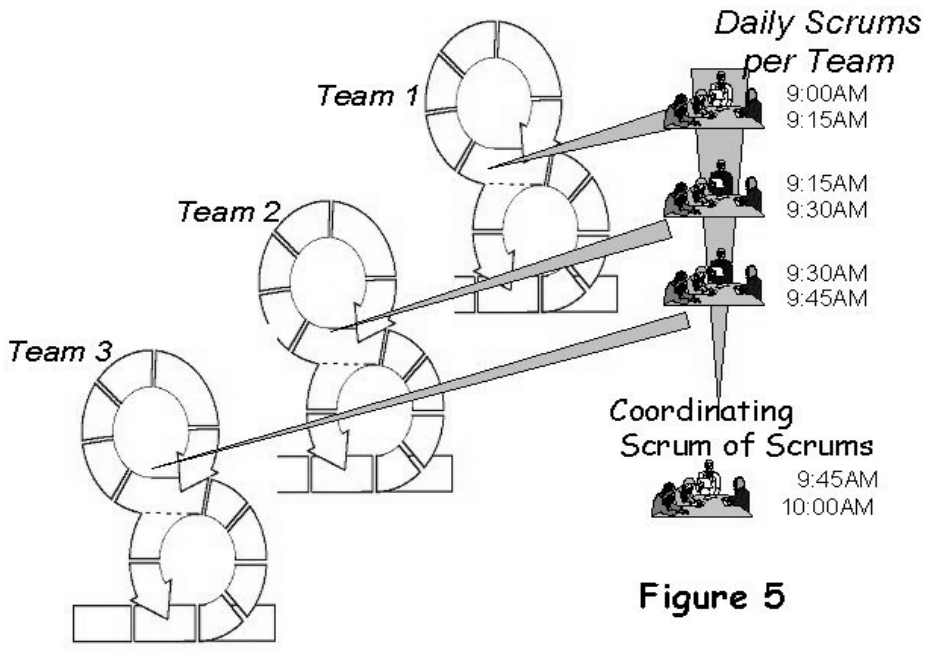


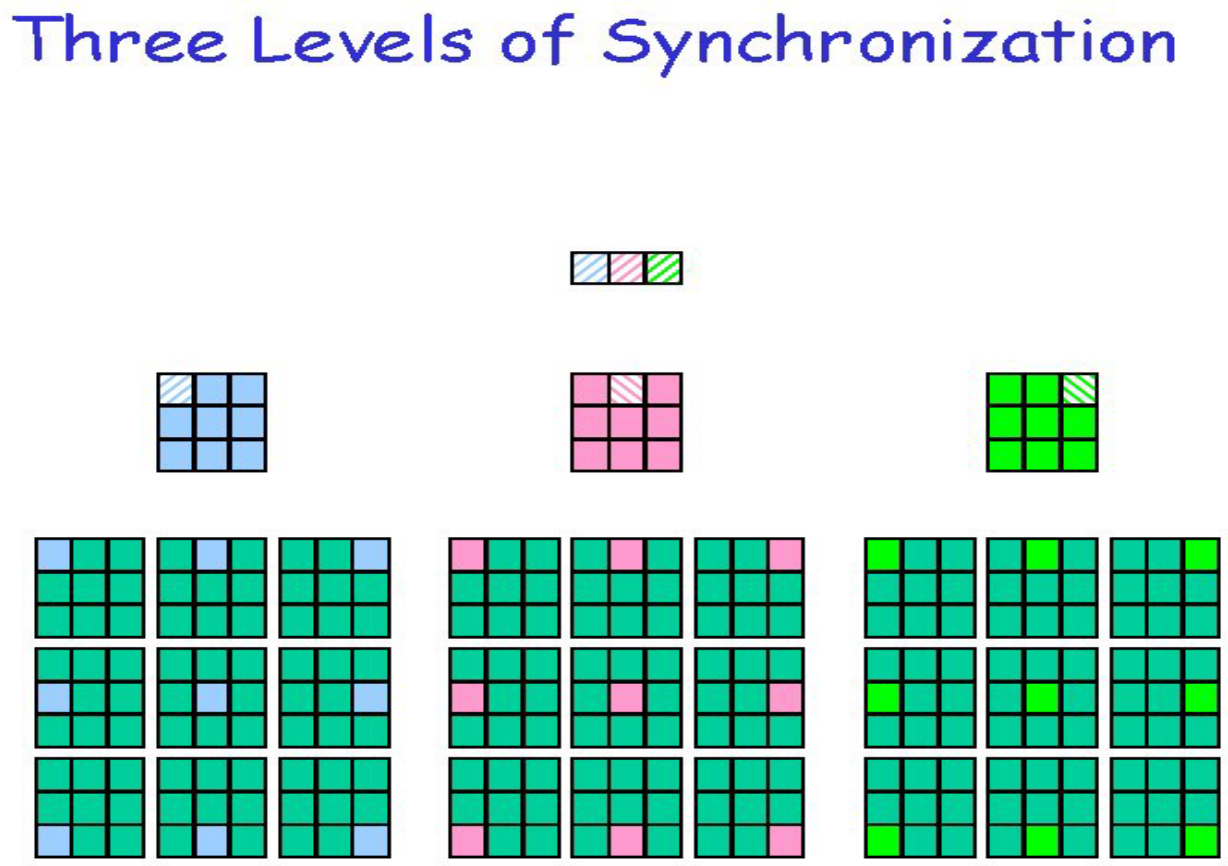
Figure 5

In Figure 5, each teams meet for their daily Scrums from 9 to 9:45 am., followed by the Scrum of Scrums at 9:45. This assumes that the ScrumMaster is used for all teams. If each team has its own ScrumMaster, all teams can meet simultaneously (say, at 9:00), and then have the synchronizing meeting shortly thereafter.

Activity ID: *10.6 Scrum of Scrums*

This idea of "Scrum of Scrum" can filter upward. To avoid having the day become too cluttered with these meetings, though, the daily Scrums further up in the hierarchy are held less frequently, but at least weekly.

The figure below provides a graphic representation of the idea of multiple levels of coordination between Scrum teams. This graphic was provided through permission of Mike Cohn of Mountain Goat Software.



Reproduced with permission from Mike Cohn, Mountain Goat Software, 2003

Phase Name: *11 End of Sprint Review*

The Sprint Review meeting is a four-hour informational meeting. During this meeting, the team presents to management, customers, users, and the Product Owner the product increment that it has built during the Sprint.

The Sprint Review provides an inspection of project progress at the end of a Sprint, every thirty calendar days. Based on the inspection, adaptations can be made to the project. The team has estimated where it will be at the end of the Sprint and set its course accordingly. At the end of the Sprint, the team presents the product increment that it has been able to build. Management, customers, users, and the Product Owner assess the product increment. They listen to the tales the team has to tell about its journey during the Sprint. They hear what went wrong and what went right. They take a fix on where they really are on their voyage of building the product and system. After all of this, they are able to make an informed decision about what to do next. In other words, they determine the best course to take in order reach their intended destination. Just like "shooting the stars" provides regularity to shipboard life, the thirty-day Sprint cycle provides a meaningful rhythm in the team's life and even in the company's life. The Sprint Review meeting happens every thirty days, and the team builds product during the other twenty-nine days.

Post-Sprint Demonstration and Meeting

Different groups of people attend the Sprint Review. Management comes to the Sprint Review to see what the team has been able to build with the resources that it has provided. Customers come to the Sprint Review to see if they like what the team has built. The Product Owner comes to the Sprint Review to see how much functionality has been built. Other engineers and developers come to the meeting to see what the team was able to do with the technology. Everyone wants to see what the team has built, what the Sprint was like, how the technology worked, what shortcuts had to be taken, what things it was able to add, and its ideas there are as to what can be done next.

During the meeting, everyone visualizes the demonstrated product functionality working in the customer or user environment. As this is visualized, consider what functionality might be added in the next Sprint. The product increment is the focal point for brainstorming. For example, someone could suggest the following after seeing the product increment demonstrated: "If we did controlled patient costs manually, we could use this right now in registration!" or "This would solve the problems that we're having tracking inventory in the districts. What would we have to do to make this work off the inventory database?" As the team demonstrates the product increment, it helps the attendees understand the weaknesses and strengths of the product increments, and the difficulties and successes it experienced pulling it together.

Subject: 11 End of Sprint Review

No one should prepare extensively for the meeting. In order to enforce this rule, PowerPoint presentations are forbidden. If the team feels that it has to spend more than two hours preparing for the meeting, then it is usually has less to show for the Sprint than it had hoped, and it is trying to obscure this fact with a fancy presentation. Sprint Review Meetings are very informal. At these meetings, what matters is the product the team has been able to create. The Sprint Review is a working meeting. Questions, observations, discussions and suggestions are allowed, and even encouraged. If a lot of give and take is needed, it should happen. Remember, though, that the meeting is informational, not critical or action-oriented. Everyone should get an understanding of the product increment, as this is the knowledge that they will need for the Sprint Planning meeting.

Activity ID: *11.1 Conduct Review*

Responsible: *ScrumMaster*

Purpose: The Sprint Review meeting is a four-hour informational meeting. During this meeting, the team presents to management, customers, users, and the Product Owner the product increment that it has built during the Sprint. The ScrumMaster is responsible for scheduling, setting up, and ensuring that all of the agenda items are covered.

Assumptions: The ScrumMaster has scheduled this meeting during the Sprint Planning meeting and the meeting is conducted on the last day of the Sprint.

Risks: This meeting is a presentation by the team that invites comments and questions from the Product Owner, customers, users, and other attendees. Make sure all of the comments are understood and captured for use in the Sprint Planning Meeting. Also, keep the tone of the meeting constructive.

Description:

The ScrumMaster is responsible for coordinating and conducting the Sprint Review meeting. The ScrumMaster meets with the team to establish the agenda and discuss how the Sprint results will be presented and by whom. The ScrumMaster sends all attendees a reminder a week before the meeting, confirming the time, date, location, attendees, and agenda.

An agenda is:

1. ScrumMaster provides review of Sprint goals, functionality chosen and functionality actually developed and to be demonstrated.

2. Team provides overview of the Sprint and functionality.

3. Team demonstrates the functionality on various workstations. Sometimes, if the audience is large, various team members demonstrate different pieces of functionality simultaneously until everyone has seen everything.

4. Meeting reconvenes and ScrumMaster facilitates a discussion of impressions and observations on what was just demonstrated.

5. ScrumMaster facilitates Sprint retrospective, a discussion of what went right and what went wrong in the Sprint, and what can be done to improve the next Sprint.

6. ScrumMaster facilitates a discussion of the impact and implications of the demonstrated functionality of the project schedule and product backlog.

7. ScrumMaster announces time and place of the Sprint Planning meeting that will initiate the next Sprint.

Activity ID: *11.1 Conduct Review*

The best presentations usually start with the ScrumMaster giving a concise overview of the Sprint. The Sprint goal and Product Backlog are compared to the actual results of the Sprint, and reasons for any discrepancies are discussed. A team member can display and review a simple product architecture diagram. The most effective architecture diagrams display both the technical and functional architecture. Previously completed technology and functions are highlighted on the diagram. Technology and functionality produced during the past Sprint are then added onto the diagram, and team members demonstrate the functionality as it is added to the diagram. For the most part, the Sprint Review meeting is held in just one place, but during the demonstration of product functionality, the meeting will often move from one workstation and office to another.

Activity ID: *11.2 Demonstrate Functionality*

Responsible: *Team*

Purpose: The team committed to a Sprint Goal and developing product functionality at the Sprint Planning meeting. Thirty calendar days later, the team is responsible for demonstrating the working functionality to the Product Owner and other interested parties. The team should spend no more than 1 hour preparing for this meeting. The only thing that actually matters is the working functionality; everything else is an artifact that led to it. The only thing that actually matters is the working functionality; everything else is an artifact that led to it.

Assumptions: The team has developed product functionality as a potentially shippable increment of the system. This means that the quality is high, both internally and externally, and that all artifacts required for the system to potentially ship have also been created. These might include user documentation, traceability maps, or performance models, depending upon the application and its intended use.

Risks: The meeting provides an opportunity to demonstrate working functionality. This is not a formal presentation that is extensively prepped and propped. Demonstrate the functionality where it was developed, on the workstations, development server, or - even better - the QA server.

Description:

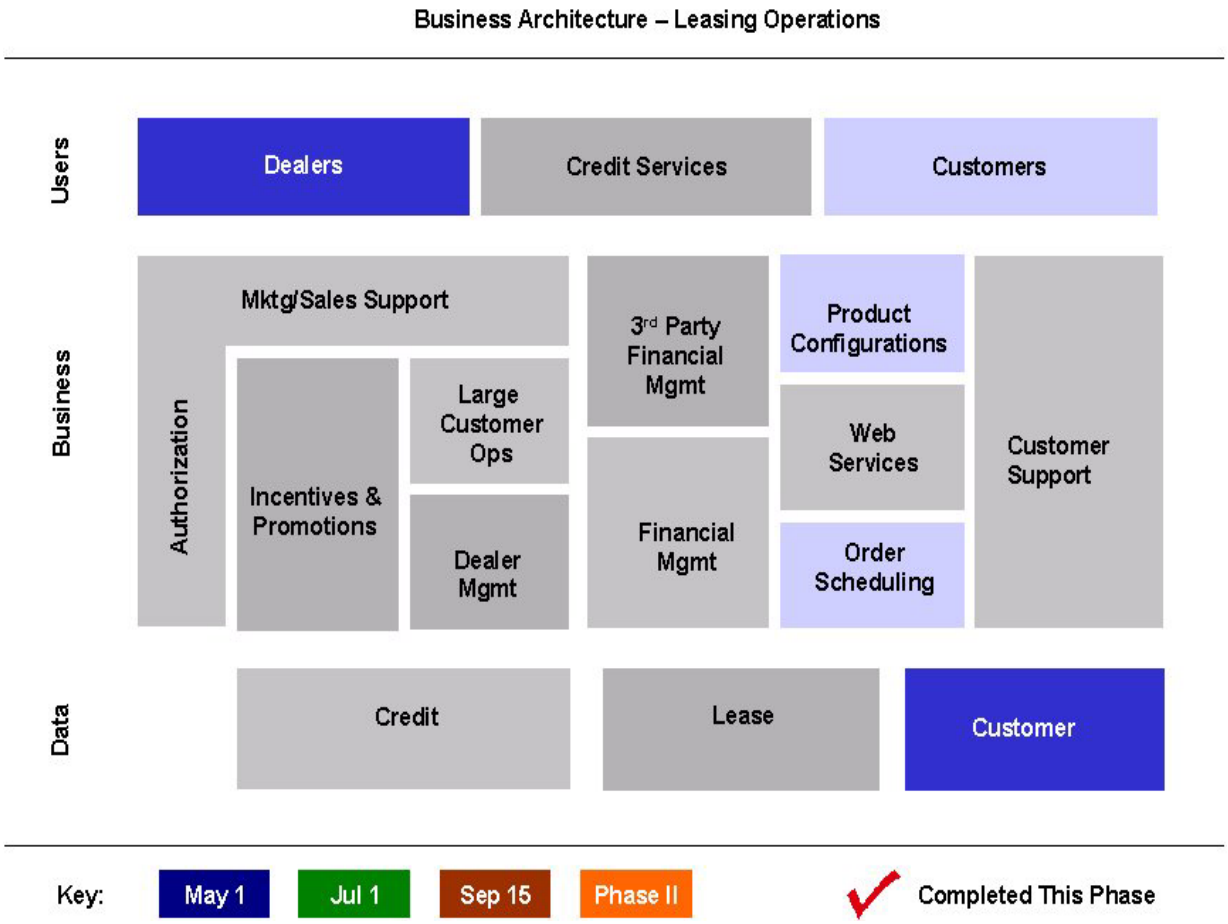
To prepare for the meeting, the team considers what attendees need to see in order to understand what has been developed during the Sprint. The team wants everyone to understand as many dimensions of the product increment as possible. What should attendees learn from this meeting? They should gain an understanding the system and technical architecture and design that holds the product together, as well as the functionality that has been built onto the architecture. They should be familiarized with the strength and weaknesses of the design and technology so they will know what limitations to be taken account of and what advantages to leverage when planning the next Sprint.

A team member can display and review a simple product architecture diagram. The most effective architecture diagrams display both the technical and functional architecture. Previously completed technology and functions are highlighted on the diagram. Technology and functionality produced during the past Sprint are then added onto the diagram, and team members demonstrate the functionality as it is added to the diagram. For the most part, the Sprint Review meeting is held in just one place, but during the demonstration of product functionality, the meeting will often move from one workstation and office to another.

As part of the meeting, the team should be prepared to discuss what it felt went well during the Sprint, and what didn't go so well. Whenever possible, have recommendations about what could be done to improve the next Sprint.
During the meeting, everyone visualizes the demonstrated product functionality working in the customer or user environment. As this is visualized, consider what functionality might be added in

the next Sprint. The product increment is the focal point for brainstorming. For example, someone could suggest the following after seeing the product increment demonstrated: "If we did controlled patient costs manually, we could use this right now in registration!" or "This would solve the problems that we're having tracking inventory in the districts. What would we have to do to make this work off the inventory database?" As the team demonstrates the product increment, it helps the attendees understand the weaknesses and strengths of the product increments, and the difficulties and successes it experienced pulling it together.

Sometimes the team uses a model of their progress through the business and systems architecture models created during the Planning phase. This is a precursor to the actual demonstration and helps the customers, stakeholders, and Product Owner understand the context and depth of what is being demonstrated. An example of a business architecture model being used for these purposes is presented below:



Activity ID: *11.3 Evaluate the Functionality*

Responsible: *Product Owner*

Purpose: The team committed to a Sprint Goal and developing product functionality at the Sprint Planning meeting. Thirty calendar days later, the team is responsible for demonstrating the working functionality to the Product Owner and other interested parties. The Product Owner is responsible for evaluating this functionality and using this understanding as a key input to formulating the next Sprint.

Assumptions: The demonstrated product functionality is potentially shippable. That is, the functionality is of acceptable quality with accompanying material that if the Product Owner chose to release or implement it, only a modest release Sprint would be required to package the functionality.

Risks: The functionality demonstrated is not of release quality. If it isn't, product backlog must be created to bring it this quality level.

Description: The team committed to a Sprint Goal and developing product functionality at the Sprint Planning meeting. The Product Owner, customers, users, and management are responsible for assessing the work of the team in relation to their expectations.

The functionality is demonstrated rather than talked about or models of it displayed. In this manner, the Product Owner and others can directly respond to the reality of the work.

Activity ID: *11.4 Manage the Release*

Responsible: ***Product Owner***

Purpose: The team has demonstrated the functionality that it was able to develop during the Sprint. The Product Backlog that described this functionality needs to be assessed so that they Product Backlog list can be made as accurate as possible.

Assumptions: The demonstrated product functionality is potentially shippable. That is, the functionality is of acceptable quality with accompanying material that if the Product Owner chose to release or implement it, only a modest release Sprint would be required to package the functionality.

Risks:

Description: If the functionality that the team demonstrated and delivered is different from the Product Backlog that it selected at the Sprint Planning meeting, the Product Backlog needs to be adjusted by the Product Owner. The following adjustments may be made:

- If some functionality was not developed, the requirements for this functionality needs to be placed back on the Product Backlog;
- If some functionality was developed with less completeness or depth than required for release and implementation, requirements for the functionality to be completely developed needs to be placed on the Product Backlog;
- If the selected technologies are not adequate to deliver the expected value, additional nonfunctional requirements need to be placed on the Product Backlog;
- If the development, QA, or production environments are inadequate or can be optimized to increase productivity, nonfunctional requirements need to be placed on the Product Backlog; and,
- If the team selected and completed additional functionality that addresses Product Backlog not initially selected for the Sprint, these items need to be removed from the Product Backlog.

As a consequence of seeing what the team was able to develop during the Sprint, the Product Manager may need to make adjustments. Possibilities include:

Activity ID: *11.4 Manage the Release*

- Restoring unfinished functionality to the Product Backlog and prioritizing it.
-. Removing functionality from the Product Backlog that the team unexpectedly completed.
- Working with the ScrumMaster to reformulate the team.
- Reprioritizing the Product Backlog to take advantage of opportunities that the demonstrated functionality present.
- Ask for a release Sprint to implement the demonstrated functionality, alone or with increments from previous Sprints.
- Choosing not to proceed further with the project and not authorizing another Sprint.
- Requesting that the project progress be sped up by authorizing additional teams to work on the Product Backlog.

The Product Manager is responsible for empirically manage plan and commitments at the end of every increment to achieve ROI by:

• Modifying requirements or priorities
• Adjusting costs
• Changing team content
• Changing release or Sprint content plans
• Call for a release

This is achieved through collaboration between business project manager, IT project manager, team, users, and other management. They always want to investigate and modify the four variables of a project to maximize business values, where these variables are the cost, date, quality, and functionality. Business value is a function of cost, quality, time, and functionality. Scrum enables value driven projects which leave the determination of the four variables tot he Product Owner throughout the project. This happens by the Product Owner only committing to one Sprint at a time, and is free to change any of the variables based on progress to date and the business value that is provided.

Activity ID: *11.5 Sprint Retrospective*

Responsible: *Team, Product Owner, ScrumMaster*

Purpose: The Scrum process works out of the box. However, as organizations understand it they often want to make modifications to make it suit their environment better. Also, when the project was formulated, staffing, tools, and environments were selected. As the project progresses, these also may require adjusting.

Assumptions: The retrospective is a collaborative process involving all members of the team, the Product Owner, and the ScrumMaster.

Risks:

Description: After every Sprint, everyone on the project takes time to reflect. What went well in the Sprint? What went not so well? What could be improved? What did we do that we want to continue doing?

When the project is started, the inputs are the Scrum process, the ScrumMaster, the Product Owner, the business domain experts, the team(s), the development and QA environments, the engineering practices and standards, and the technologies. The basis of Scrum is inspect and adapt. This Sprint Retrospective is the time to inspect all of the above items, and anything else that affects the project, and to make adjustments so that the team can do better. Nothing is out of bounds in this discussion, which is intended to be open and fruitful. The ScrumMaster has had excellent visibility into the project for the last Sprint and is expected to have the most suggestions.

Activity ID: *11.6 Project and Sprint Reporting*

Responsible: *Product Owner, ScrumMaster*

Purpose: When the project was funded, expectations regarding costs, benefits, releases, and progress were established. These expectations are revisited at the end of every Sprint and management reports tracking the project and discussing any changes or modifications are detailed in management reports.

Assumptions: The people to whom the project is reporting are aware that Scrum and iterative, incremental development are being used.

Risks: Most management reporting for projects talks about work, percent of work done, and milestones. Scrum reporting talks about functionality completed, progress toward being able to release functionality, and costs versus benefits. The transition from one reporting structure to another may not be easy. In some cases, prepare both types of reports so that management can relate the new type of reports to previous reports. Report in person whenever possible.

Description: The project managers report periodically on progress. These reports can include such standard verbiage as issues, risks, assessment of progress, and discussion of changes. These reports should resonate with the expectations and metrics established during the planning phase. The following were recommended to be established at that time:

- Release Schedule - planning out the various releases and their goals and expected impact;

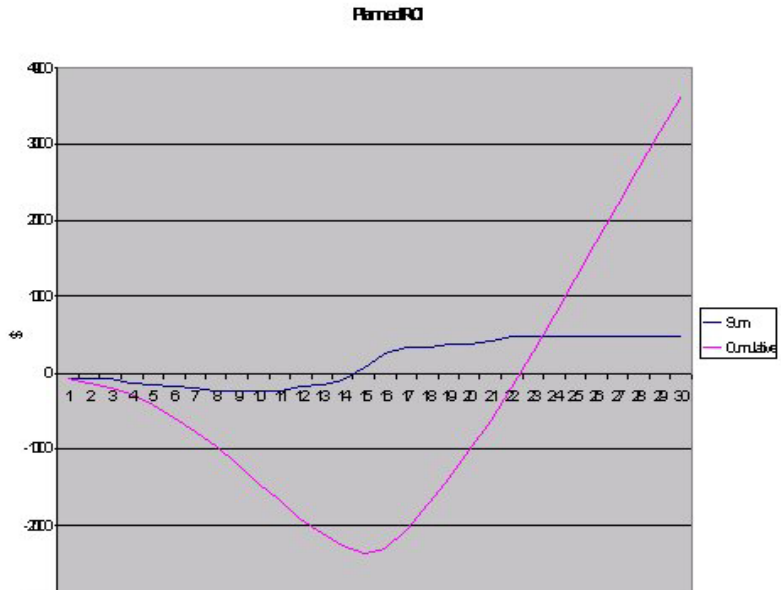
- Product Backlog - what functionality will probably be build and delivered when, and as part of what release; and,

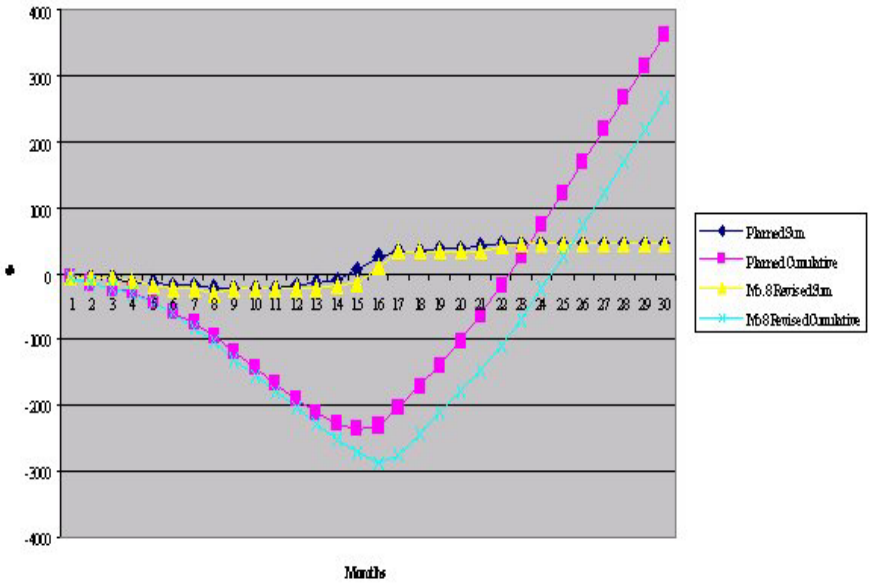
- ROI analysis based on the cost/benefit and expense/revenue spreadsheets.

Reporting usually occurs at two levels. A summary level is appropriate for those who wish to be aware of the projects overall progress. A detailed level is appropriate for management that is wants to know both the overall progress and why deviations are occurring.

The basis of the summary reporting is a projection of the cost/benefit models prepared during the planning phase across time.

For instance, the figure below charts the expected costs and benefits, both as they occur and cumulatively. Upper line is monthly summary of costs and benefits
Lower line is cumulative by month, showing break even expectation at month 22. This expectation was set with management during the planning phase as part of acquiring funding for the project.





The next figure shows the impact of the release planned at month 5, at which point benefits should have started to accrue, being delayed until month 8 (with subsequent delay of other planned releases.

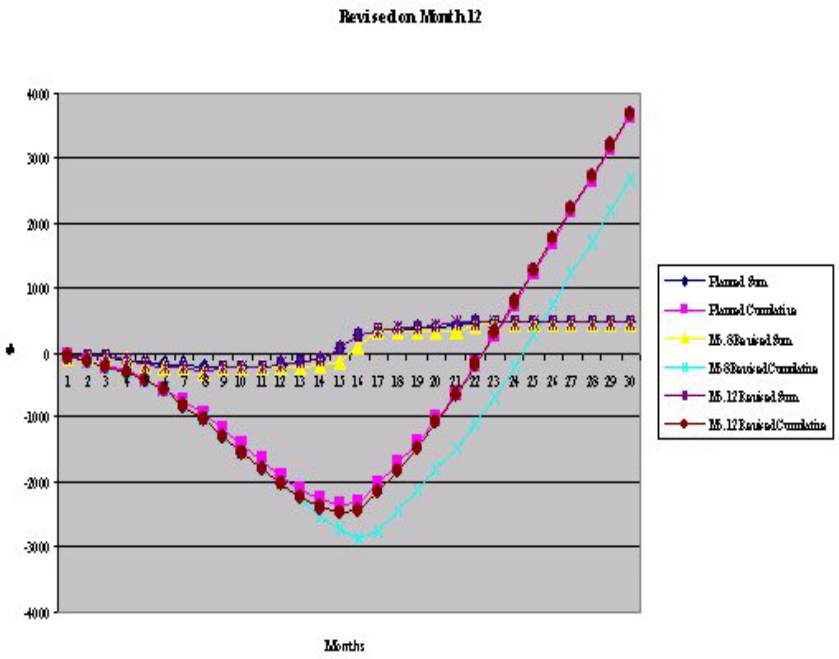
In this figure, there are two sets of lines. The upper set tracks monthly expenses and benefits (run rates). The upper line is the planned, the bottom line the actual with projections.

The lower of lines tracks the cumulative cost/benefit curve. The upper of the two lines tracks the plan, the lower of the two lines tracks the actual and projected.

In the plan vs actual report, breakeven has moved out from the 22nd month to the 25th month.

In this report, the Product Owner ascribes the later release to the lack of business domain expertise, which will be resolved with new team members in the upcoming Sprints.

Activity ID: *11.6 Project and Sprint Reporting*



In the third report, the Product Owner is able to demonstrate that the break even point can now be projected as returning to the 22nd month due to greater productivity since the addition of the domain experts at the end of the fifth month. In the lower set of lines representing a plotting of cumulative cost/ benefits, the projection at the end of the twelfth month shows the pulling together of the cumulative cost/ benefits due to this increased productivity.

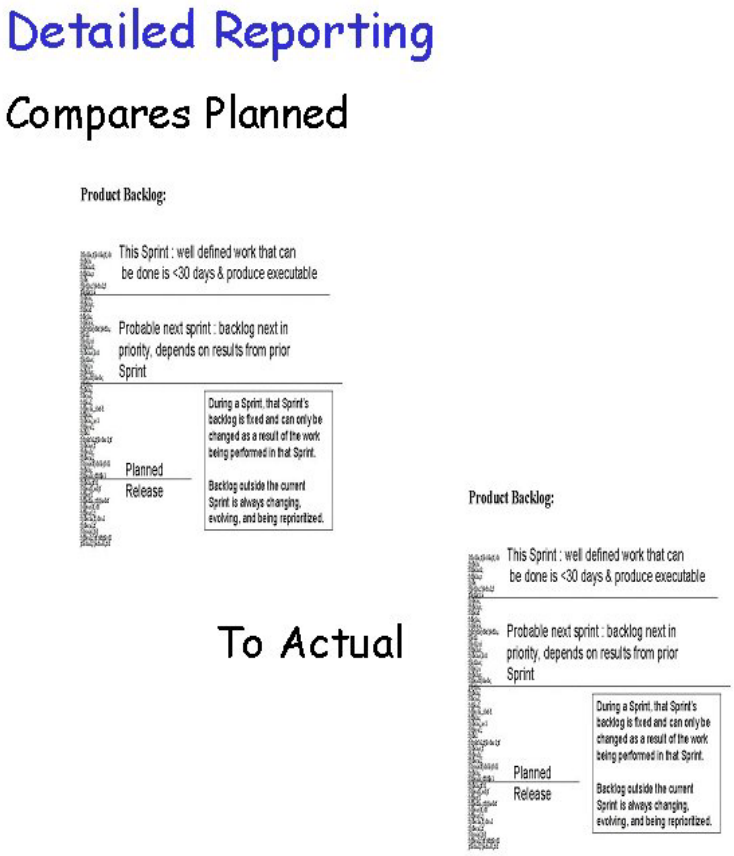
Detailed reporting is done at the end of each Sprint to management that tracks the project's progress more closely. The most appropriate reporting in Scrum is the frequent end of Sprint evaluation of the functionality that the team built that Sprint. The reporting shows how that functionality compares historically with expectations and lays out an action plan if that progress is inadequate or needs revision.

At the start of a project, the Product Backlog was prepared. The Product Backlog consisted of the functional and nonfunctional requirements prioritized into a list. At specified dates, releases were indicated. The releases were described in terms of goals and expectations to be met by the delivery of the functionality. Sprints, with intermediate goals, were predicted that would lead to the delivery of the release.

At the end of every Sprint, the Product Backlog is revised based on actual functionality demonstrated. The incomplete functionality is returned to the Product Backlog.

Activity ID: *11.6 Project and Sprint Reporting*

The Product Backlog has also changed due to changes in the business environment. What was important at the start of the Sprint may be more or less important at the end of the Sprint. A new Product Backlog is constructed now indicating the new Product Backlog contents and priorities, along with any changes to release dates and contents, as indicated by where the release line is inserted into the Product Backlog.



Accompanying the prior and current Product Backlogs is ana analysis report, indicating all of the major and minor changes, as well as the cause for the changes. Changes made to the project that will affect its productivity are also indicated, such as the addition of domain expertise in the above examples.

A series of these comparisons represents the project history from a management reporting aspect.

At the core of these detailed reports, are the following:

- ROI Analysis – a comparison of ROI as documented by the updated ROI gauges compared to the ROI projected at the start of the project and in the previous status report.

- Product Backlog analysis – a discussion of the Product Backlog presented at the previous status report with the current backlog. The discussion includes why priorities have changed and why release dates have changed. Early implementation reasoning is presented.

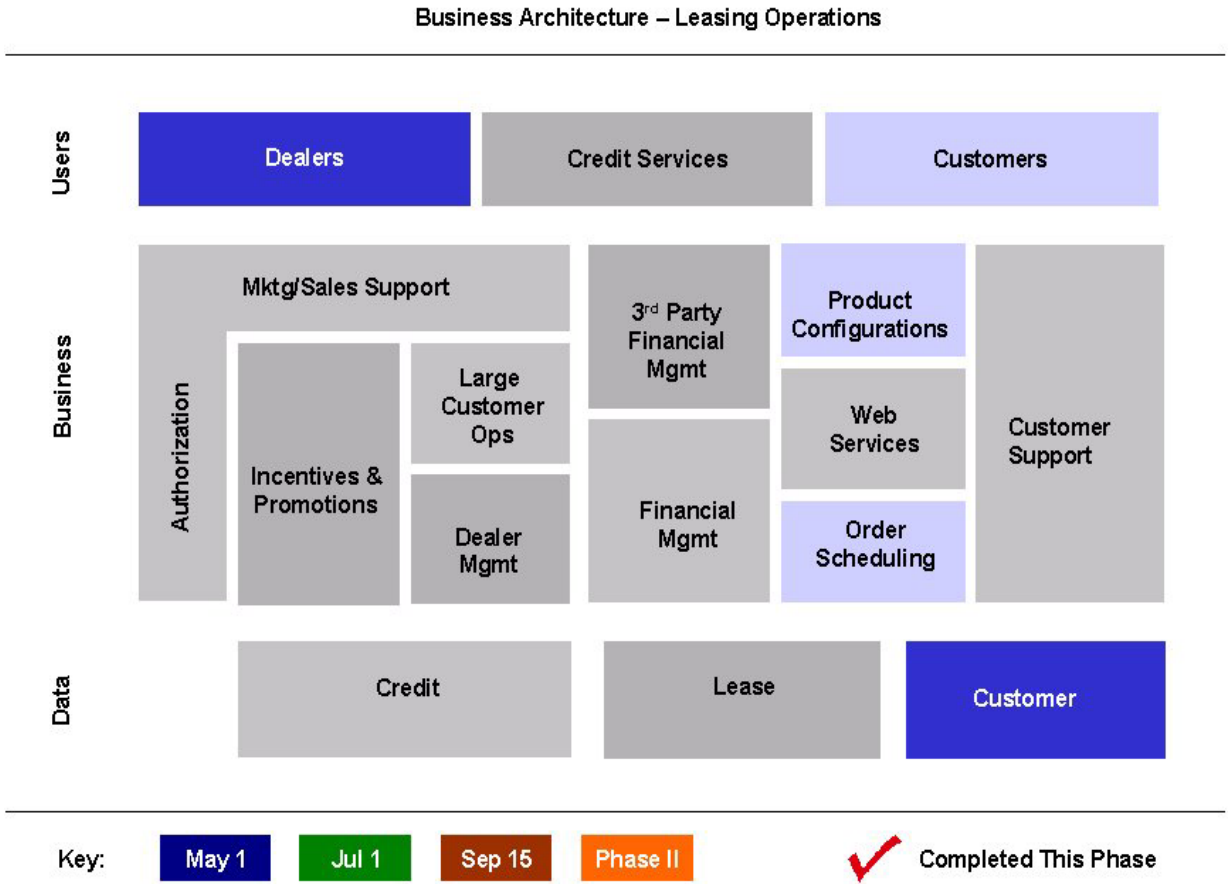
- Recommendations – based on the ROI and Product Backlog analysis, what changes are recommended? Since Scrum is iterative, costs can be increased or decreased by adding or reducing teams. Changes in release schedules may be recommended for competitive reasons.

Reporting should be based on the Product Backlog and changes made to its content, priorities, Sprint schedules and release schedules from initial expectations and the previous Sprint reports. Burndown charts should be used to demonstrate progress against expectations, demonstrating probable trends.

Activity ID: *11.6 Project and Sprint Reporting*

Progress reporting is also sometimes augmented with updates to the business and system architecture models that were constructed during the Planning phase. This provides visual details that supplement the ROI and Product Backlog reporting and allows customers to visually track project progress.

An example of progress tracking after the first Sprint of a project is provided below. Progress is indicated on the business architecture model through color coding.



Activity ID: *11.7 Attend the Sprint Review*

Responsible: ***Chickens***

Purpose: People who are not part of the team but who want to be aware of the project status at the end of the Sprint are invited to attend the end of Sprint Review. These people are called "chickens" to differentiate them from the "pigs" on the team.

Assumptions: The ScrumMaster has provided a mechanism for anyone interested to learn the location and time of the Sprint Review meeting. Those interested include all stakeholders in the project, including customers, users, sources of funding, and other technology parties.

Risks: The meeting gets out of hand because of too many viewpoints or conflicting agenda.

Description: The following rules apply to the Chickens at the Sprint Review and are applied by the ScrumMaster:

1. Chickens are not allowed to interrupt the presentations;
2. Chickens are free to voice any comments, observations, or critiques regarding the increment of potentially shippable product functionality between presentations;
3. The chickens can identify functionality that wasn't delivered or wasn't delivered as expected and request that such functionality be placed in the Product Backlog for prioritization;
4. The chickens can identify any new functionality that occurs to them as they view the presentation and request that the functionality be added to the Product Backlog for prioritization;
3. If too many chickens attend the meeting, the ScrumMaster may limit attendance so the meeting can remain orderly and focused.

Phase Name: ***12 Create a Release***

Under Construction

Activity ID: ***12.1 Create Product Backlog***

Responsible: ***Product Owner***

Purpose:

Under Construction

Assumptions:

Risks:

Description:

Activity ID: ***12.2 Initiate Sprints to Build Release***

Responsible: ***ScrumMaster***

Purpose:

Under Construction

Assumptions:

Risks:

Description:

Subject: ***Roles and Responsibilities - Product Owner***

Scrum introduces the concept of workload management to systems development. Workload management involves controlling development of functionality and release dates in order to optimize the value to the organization of the system being developed. This is different from work management, in which the specific tasks involved in building a system are directed.

Scrum makes workload management possible through iterative, incremental development using thirty day iterations called Sprints. An increment of functionality is ready by the end of every Sprint. The term "ready" here means potentially shippable or able to be implemented. "Ready" means complete, including user documentation, and having been tested fully.

Traditional development methodologies fully analyze and design a system before coding it. Testing usually follows the coding. It is not until the very end of the project that the system can be implemented. The opportunities for managing this workload to optimize value are limited and are usually not even considerable. However, Scrum makes it possible to perform analysis, design, testing, coding and documentation in every Sprint. This provides management with many opportunities to do the following:

1. arrange the sequence in which functionality is iteratively developed so that the most valuable functionality is built first;
2. continue to rearrange the sequence of functionality development as the project progresses and business priorities change; and,
3. group increments of functionality into more frequent releases, allowing the business to realize early and frequent benefits.

Consider a system that will bring the organization $1,000,000 dollars in benefits in the first two years after its implementation. Using traditional methods, the system would take one year to develop, and its development would cost $400,000. Agile processes let us develop and implement the system's functionality selectively and incrementally by doing the following:

1. listing the functionality of the system, with more attention to the highest value and priority functionality;
2. dividing the functionality list into two releases, the first estimated to be ready six month after development begins;
3. using iterative, incremental development to complete the first release within six months for $200,000;
4. allowing benefits worth $800,000 to begin accruing after just six months, with the functionality that will deliver the remaining value scheduled to be developed during the second Sprint; and,
5. permitting the second implementation to be deferred if it is not deemed cost effective and the benefits of the first implementation are deemed sufficient—for example, if the development cost of $200k for the less valuable functionality would generate only $200k in benefits.

Workload management such as is described in this example is made possible by agile processes. In this case, the customer had an opportunity to realize $800k in benefits six months earlier than would otherwise have been possible. The customer also had the opportunity to choose not to spend an additional $200k for break-even functionality. The time and effort that would have gone into the second Sprint could instead be allocated to other higher priority projects. The benefits of multiple releases are somewhat offset by implementation costs.

Strategic and competitive systems are able to gain marketplace advantage through such incremental strategies. Imagine that your competition uses traditional development approaches to prepare a single new release or business capability, but your organization uses agile processes to produce early and repeated competitive advantages. If this is the case, then your organization is able to capture the advantage more effectively and thoroughly.

An additional benefit of workload management is inventory reduction. As in manufacturing, unfinished "raw goods" software inventory is an undesirable cost. It may need to be reworked if it has defects. It may never even be used if production costs are too high or demand for the software evaporates. Yet traditional development methodologies amass huge inventories of analysis, design, and coding artifacts even as business changes render them obsolete. The agile approach minimizes the extent to which an organization accumulates such artifacts. Only those artifacts that are necessary to build each iteration's increment of functionality—the highest priority functionality—are built.

This role of workload management is a key new role afforded by agile processes. This role is referred to as "the Customer" in Extreme Programming and as "the Product Owner" in Scrum. These roles have responsibilities that enable the realization of the benefits of workload management. The Product Owner executes this role through active management of an inventory called Product Backlog. The Customer executes this role through active management of an inventory of Product Backlog.

Let's look more closely at Product Backlog. Product Backlog is a simple list of requirements for the system. Each item on the list is a single line in length. Functional requirements, such as "the ability to calculate available credit," are listed along with nonfunctional requirements, such as "the capacity to handle up to 100,000 simultaneous transactions with sub-second response time." Product Backlog is often maintained in spreadsheet format so that it can be easily manipulated and interpreted.

The Product Backlog is a prioritized list. Items at the top of the list are those that will deliver the most business value. Business priorities can change over the course of the project, and consequently the order of the list can change as well. Dependent functionality, or functionality that is required to support highest priority backlog, is of an even higher priority. Each backlog item is a one-line description of the requirement. An estimate of how it will take developers to turn the functionality into an increment of potentially shippable product is included in each backlog item.

The Product Owner doesn't have to specify all the details of every entry in the Product Backlog. The Product Owner extracts requirements from the systems plan, focusing on the highest priority Product Backlog first. At first, the Product Owner needs to list only as much Product Backlog as is needed to drive the first probable release. The lower priority functionality can be itemized and delivered only when it is deemed to be the highest priority available functionality. Even then, its development may be deferred if it costs more than it is worth.

Subject: Roles and Responsibilities - Product Owner

Only one person is responsible for managing and controlling the Product Backlog. This person is referred to as the Product Owner. For commercial development, the Product Owner may be the product manager. For in-house development efforts, the Product Owner could be the project manager, or the user department manager. This is the person who is officially responsible for the project. This person maintains the Product Backlog and ensures that it is visible to everyone. Everyone knows what items have the highest priority and so everyone knows what will be worked on.

The Product Owner is one person, not a committee. Committees may exist that advise or influence this person, but any person or body of people wanting an item's priority changed has to convince the Product Owner to make the change. Organizations have many ways of setting priorities and requirements. These practices will be influenced by Scrum across time, particularly through the meeting that reviews product increments (Sprint Review). The practice that Scrum adds is that only one person is responsible for maintaining and sustaining the content and priority of a single Product Backlog. Otherwise, multiple conflicting lists flourish and the Scrum teams don't know which list to listen to. Without a single Product Owner, far too much floundering, spin, contention, and frustration will result.

Subject: ***Roles and Responsibilities - ScrumMaster***

A new management role introduced by Scrum is the ScrumMaster. The ScrumMaster is responsible for ensuring that Scrum values, practices, and rules are enacted and enforced. The ScrumMaster represents management to the team and represents the team to management. At the Daily Scrum, the ScrumMaster listens closely to what each team member reports. He or she compares what progress has been made to what was progress was expected, based on Sprint goals and predictions made during the previous Daily Scrum. For example, if someone has been working on a trivial task for three days, it is likely that team member needs help. The ScrumMaster tries to gauge the velocity of the team: is it stuck, is it floundering, is it making progress? If the team needs help, the ScrumMaster meets with it to see what he or she can do to help.

The ScrumMaster works with the customers and management to identify and works with the customers and management to select a Product Owner. The ScrumMaster works with management to form Scrum teams. The ScrumMaster then works with the Product Owner and the Scrum teams to create Product Backlog for a Sprint. The ScrumMaster works with the Scrum teams to plan and initiate the Sprint. During the Sprint, the ScrumMaster conducts all Daily Scrums, and is responsible for ensuring that impediments are promptly removed and decisions are promptly made. The ScrumMaster is also responsible for working with management to gauge progress and reducing backlog.

The Team Leader, Project Leader, or Project Manager often assume the ScrumMaster role. Scrum provides this person with the structure to effectively carry out Scrum's new way of building systems. If it's likely that many impediment will have to be initially removed, this position may be filled by a senior manager or a Scrum consultant.

How does the ScrumMaster keep the team working at the highest possible level of productivity? The ScrumMaster does so primarily by making decisions and removing impediments. When decisions need to be made in the Daily Scrum, the ScrumMaster is responsible for making the decisions immediately, even with incomplete information. I've found that it's usually better to proceed with some decision than no decision. The decision can always be reversed later, but in the meantime, the team can continue working. As for impediments, the ScrumMaster either personally removes them or causes them to be removed as soon as possible. When the ScrumMaster does the latter, he or she makes visible to the organization a policy, procedure, structure, or facility that is hurting productivity rather than helping

Successful ScrumMasters have certain personality traits. He or she is usually focused and determined to do whatever is necessary for their Scrum teams. Some people aren't appropriate as ScrumMasters. They aren't comfortable being that visible and taking that much initiative. Removing impediments requires determination and stubbornness.

Subject: ***Roles and Responsibilities - Team***

The Development Team manages the work during each Sprint The Product Owner indicates what functionality most needs to be developed. The Development Team identifies and organizes the tasks and work necessary to the results of the Sprint into a potentially shippable product increment. Collaborating with the Product Owner, the Development Team determines how much priority functionality it believes it can cover in the next Sprint.

Scrum work management is a radical shift from traditional Project Management Institute priorities. These practices call for a project manager to predict and plan all of the work as well as to assign it to individuals, track its completion, and make any necessary adjustments along the way. Scrum work management instead follows modern lean manufacturing practices and engineered process controls used in complex development environments. Scrum teams have these characteristics:

1. They are cross-functional, containing all of the technical and business domain expertise to take full responsibility for moving from requirements forward to working product functionality.
2. They are limited in size in order to maximize the speed, content, accuracy, and bandwidth of communications. Team size is seven plus or minus two people. When there are multiple teams, the teams get together to synchronize their work on a daily basis.
3. They are authorized to organize themselves, to divide and assign work amongst themselves.
4. They are enabled to let specific tasks require to create an increment of functionality emerge as the Sprint progresses; they are expected to be able to make perfection predictions.

For the duration of the Sprint, the Team has the authority to manage itself. Its main goal is to do the best that it can. Applying the technology to the requirements, the Team analyzes, designs, codes, and tests. At the end of the Sprint, the Team demonstrates what it has accomplished. The Team uses workstations to show the Product Owner the functionality it has created. Only real working functionality count; interim artifacts such as models do not count.

Sometimes the Team does less than it has predicted it would be able to. Sometimes the Team implements the selected requirements even more deeply than it had expected it could. Regardless, the Team does the best that it can. For one Sprint, the Team alone wrestles functionality from complex, sometimes unstable technology and from often-changing business requirements.

To many, it may seem risky and even foolhardy to trust the Team to plan and execute its own work. However, this type of agile development has been successfully used in literally thousands of projects. Two types of productivity result. Firstly, nobody has to tell the team what to do, and then keep the plan up to date as changes are required. Secondly, the Team works more effectively without having to rely on external authority for any changes.

The U.S. Marine Corp. uses an approach similar to agile processes for battle situations. In *Corps Business*[1] General Charles C. Krulak , the Thirty-first Commandant of the USMC, describes the new "three block war" that the corps faces today – " Marines may confront the entire gamut of tactical

challenges within the narrow confines of three continuous blocks." To prepare the marines, the actual fighters, for this situation, the USMC both trains everyone extensively in all potential skills and situations that can be conceived, and then advises the marines on the context, mission, goals, and risks of every situation before they are sent in. But, from then on, the marines are on their own, making their own decisions. Their officers provide as much tactical information as possible, but the ultimate decisions come from the soldiers. As General Krulak says, "on the complex, asymmetrical battlefields of the twenty-first century, effective decentralized control and execution will be essential to mission success" (*Corps Business*, David H. Freedman, HarperCollins Publishers, 2000). This same type of decentralized control and execution by agile teams is required to successfully cope with the complex changing requirements and complex unstable technology required for today's successful systems. These teams manage themselves based on their skills and understanding of the technical and business domains.