First Edition

Scrum Intro
A fit for you?

Kevin O’Mara, O’Mara Consulting Associates

The Problem

1

For decades, software projects were delivered late, over budget, and often failed to meet the needs of customers.

Early Software Development

Waterfall Process

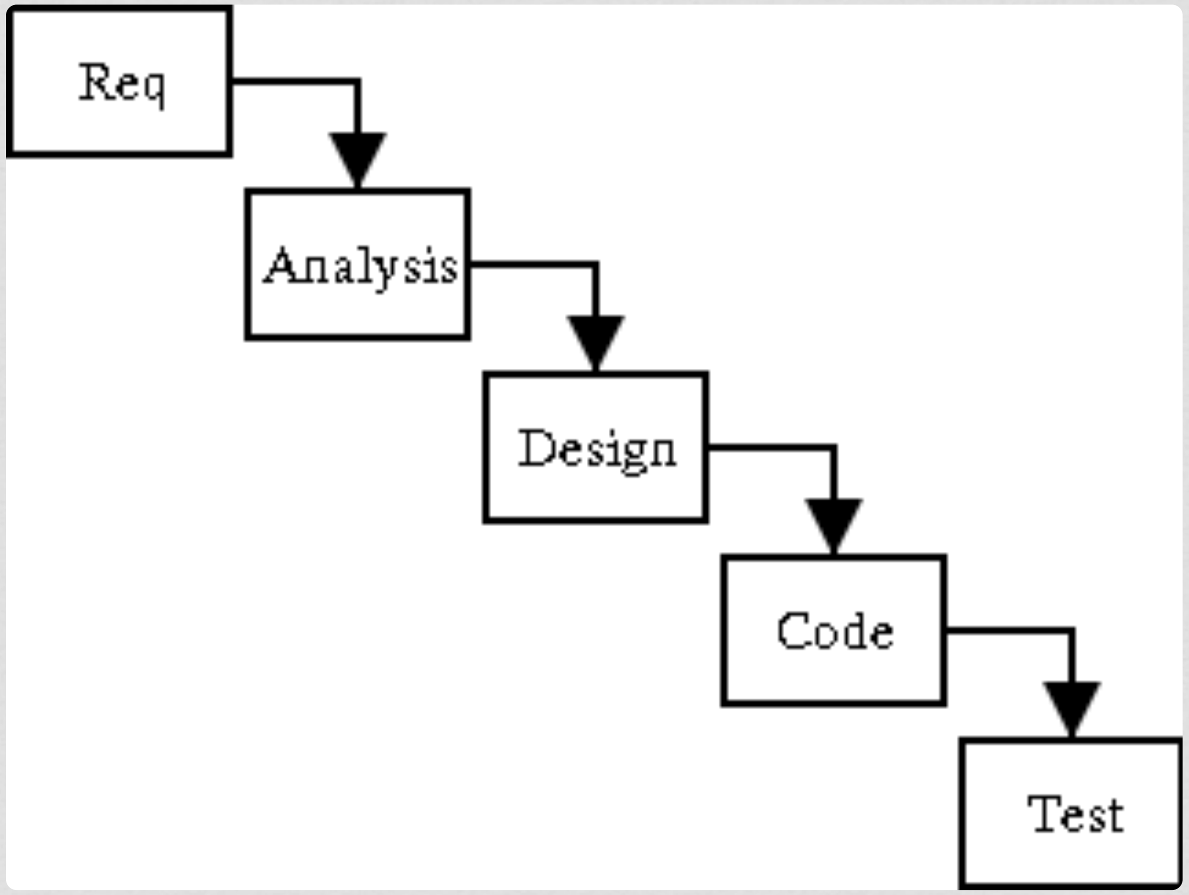
Software development is a creative process - but a process nonetheless. To realize any type of product, the supporting process is comprised of activities in five phases:

1. Requirements
2. Analysis
3. Design
4. Implementation
5. Test & Evaluation

When you observe the graphical representation to the right, it appears product development “falls” from one phase to the next, like a waterfall. The first formal description of the **waterfall model** is often cited as a 1970 article by Winston W. Royce, though Royce did not use the term "waterfall" in this article.

Royce presented this model as an example of a flawed, non-working model. This is how the term is generally used in writing about software development—to describe a critical view of a commonly used software practice.

Gallery 1.1 The Waterfall Process



The waterfall Process gets its name from the appearance of “falling” from one phase to the next.

Strengths

While the term waterfall is frequently used as a pejorative, it has persisted as a valid software development methodology for decades because it is a good fit for contract law and government procurement policies - though its detractors might claim its weaknesses negate those strengths.

A contract is a legally enforceable promise or undertaking that something will or will not occur, given for consideration. For example, "I'll deliver the new website by March 31 (the promise) for $10,000 (the consideration). Among other things, a good contract defines what constitutes "new website" - which maps nicely to Requirements in the waterfall process.

Whether a contract exists between government and/or corporate entities, departments in an organization, or boss and subordinate, a good contract sets clear expectations that are mutually understood and agreed upon. Stakeholders on both sides have legitimate needs to know "what am I going to get/deliver", "when am I going to get/deliver it", and "how much am I going to pay/receive for it".

Some entities - especially governmental, have software procurement policies that define in excruciating detail "uniform requirements for software development and documentation". For example, the US Department of Defense codified MIL-STD-2167 in 1985. That policy references 5 additional MIL-STDs that in turn define the structure and format of dozens of Specification documents - e.g., Software Requirements, Test Plan, Installation Plan, Operation Manual, and so on. Whew!

Procurement Policies are almost always written or approved by attorneys and designed for inclusion by reference into a contract. In practice, they mimic the waterfall process.

Weaknesses

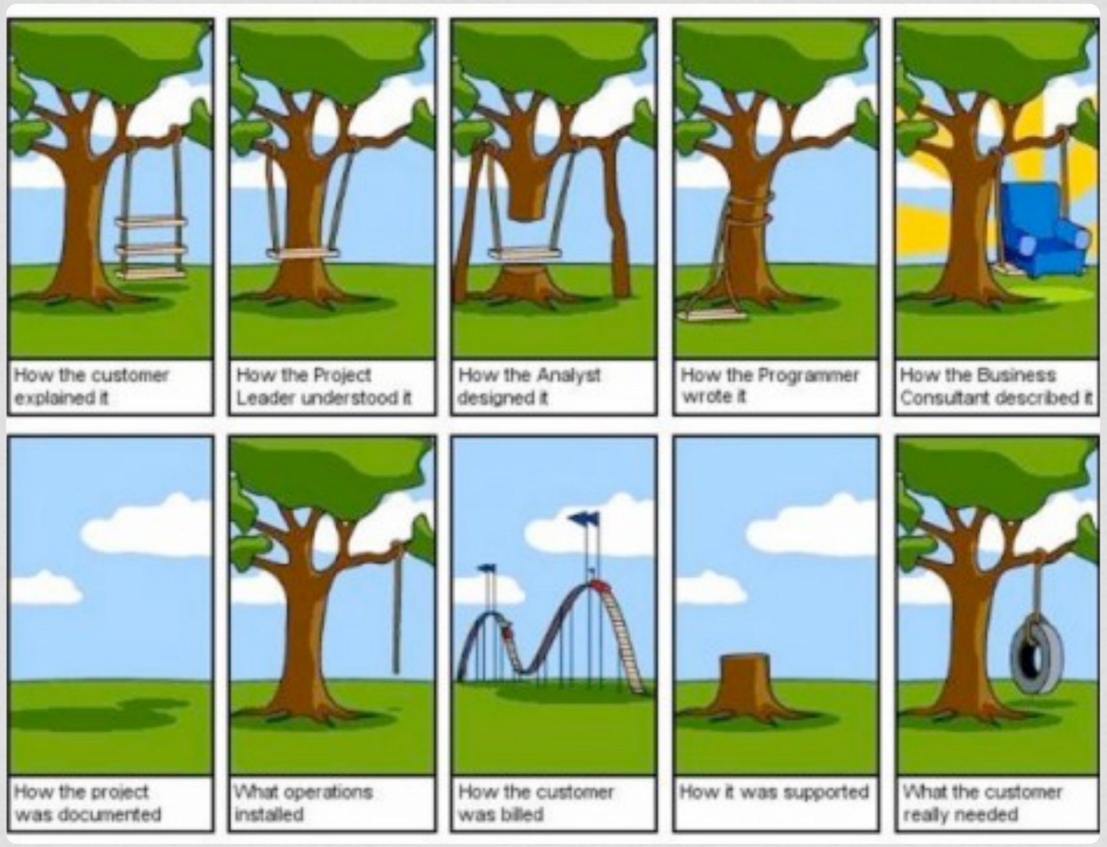
The waterfall model originates in the manufacturing and construction industries: highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware-oriented model was simply adapted for software development. But software is infinitely changeable.

While there are many problems throughout the phases of the waterfall process, the biggest problems are two implicit assumptions about Requirements:

- Requirements are well understood
- Requirements don't change

The challenge of understanding is one of complexity. Software is dynamic - the "output" might change dramatically if even one of the "inputs" changes slightly. Even a simple product has dozens of inputs - the difficulty of accurately describing this dynamic behavior in a static Software Requirements Specification is immense - some would say insurmountable.

GALLERY 1.2 Requirements Specification Humor


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Compounding this are limits of human comprehension. Even if the description is perfect, two people reading the same specification will see different "visions" in their minds. In reference to our earlier statement that "a good contract sets clear expectations that are mutually understood and agreed upon" - there is essentially a zero probability the two parties have a mutual understanding. So what did they agree upon?

The other challenge is Requirements do change over time. Competitors introduce new features, new laws and regulations are introduced, business processes change, and so on. Even if we could write perfect Requirements Specifications that all parties understand perfectly, there is a high probability the true business need represented by the specification will be different by the time the product is delivered.

Waterfall and Iterative Development

Waterfall's detractors speak as if the phases are perfectly sequential, and cite massive failures like the $1.5B write-off of the FAA's Air Traffic Control system in 2002. With the exception of the very, very early days of software development and inane policies dictated by powerful bureaucrats, nobody does pure waterfall.

There are a number of waterfall derivatives (Spiral Model, Rational Unified Process, etc.) that focus on delivering in multiple iterations to reduce the complexity of each sub-delivery and shorten the period of time during which external forces may change the requirements.

The Solution - Scrum

2

Many of the ideas in Scrum were first apprehended by Jeff Sutherland as he read the writings of Hirotaka Takeuchi and Ikujiro Nonaka in the mid-1980s.

Agile Development

Scrum is considered an Agile Methodology.

Definition

Agile software development is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle.

The Agile Manifesto introduced the term in 2001.

Types of Agile Methodologies

While not discussed in detail here, there are a number of Agile approaches that have garnered favor among product developers. Among the more popular are:

- Scrum
- Extreme Programming
- Dynamic System Development Method
- Kanban
- Lean

What is Scrum?

Origin of Scrum

The term Scrum was coined by Takeuchi and Nonaka in a 1986 paper, “The New New Product Development Game” published in the Harvard Business Review. In this paper they described a holistic approach in which the development phases overlap and where project teams are made up of small cross-functional teams. They compared these high performing teams that work together towards a common goal to the Scrum formation in rugby.

Much of the research behind Scrum and other Agile methodologies is derived from work done in Japan in the later half of the 20th Century, notably the **Toyota Production System**. Of particular relevance is the contrast between Defined and Empirical Process Control.

The defined process control model requires that every piece of work be completely understood. Given a well-defined set of inputs, the same outputs are generated every time. A defined process can be started and allowed to run until completion, with the same results every time.

The empirical model of process control provides and exercises control through frequent inspection and adaptation for processes that are imperfectly defined and generate unpredictable and unrepeatable outputs.

For many years software development methodologies have been based on the defined process control model. But software development isn’t a process that generates the same output every time given a certain input.

Jeff Sutherland created Scrum in 1993 while working at Easel, an Innovator in **Object Oriented** technology. Scrum is based on the Empirical Process Control model.

Definition of Scrum

In An Overview of Scrum by Mike Cohn, the author defines Scrum as an **agile** process that allows us to focus on delivering the highest business value in the shortest time. It allows us to rapidly and repeatedly inspect actual working software (every two weeks to one month). The business sets the priorities. Teams self-organize to determine the best way to deliver the highest priority features. Every two weeks to a month the team delivers a “potentially shippable product”.

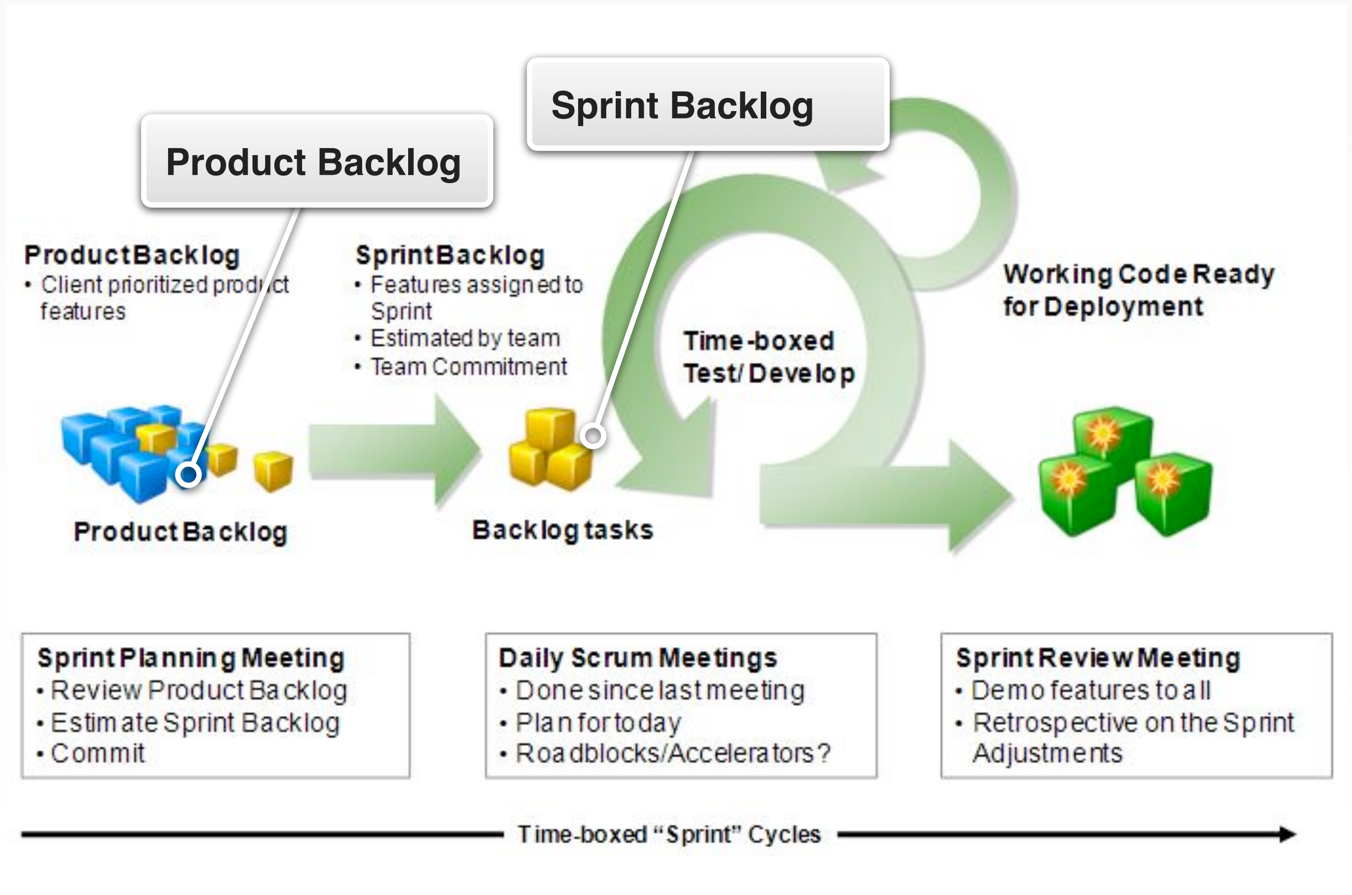
Scrum is a simple “inspect and adapt” framework that has three roles, four ceremonies, and four artifacts designed to deliver working software in 2-4 week iterations known as Sprints.

• **Roles**: Product Owner, ScrumMaster, Team.

• **Ceremonies**: Sprint Planning, Sprint Review, Sprint Retrospective, and Daily Scrum Meeting

• **Artifacts**: Product Backlog, Sprint Backlog, Impediments Backlog and Burndown Chart

FIGURE 2.1 Scrum Snapshot



Scrum Roles

There are three roles in Scrum:

- Product Owner
- ScrumMaster
- Team

Product Owner

The Product Owner is the one person responsible for a project's success - often described as “the single, wringe-able neck. The Product Owner leads the development effort by conveying his or her vision to the team, outlining work in the scrum backlog, and prioritizing it based on business value. Primary responsibilities include:

- Define the features of the product, decide release dates and content
- Profitability of the product (ROI)
- Prioritize features according to business value
- Add, change, delete features and priority every iteration
- Accept or reject work results

Most of the responsibilities are manifested in the Product Backlog.

ScrumMaster

The ScrumMaster serves as a facilitator for both the Product Owner and the Team. He or she has no management authority and may never commit to work on behalf of the team.

In Scrum, the ScrumMaster is an arduous role and demands a distinct personality type to succeed. The best ScrumMasters are real team players, who receive as much satisfaction from facilitating others' success as their own. They must also be comfortable surrendering control to the Product Owner and team. For those two reasons, traditional project managers don't usually make great ScrumMasters. Primary responsibilities include:

- Ensure the team is fully functional and productive
- Enable close cooperation across all roles and functions
- Remove impediments
- Shield the team from external interference

- Ensures the process is understood and followed. Issues invitations to Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective meetings

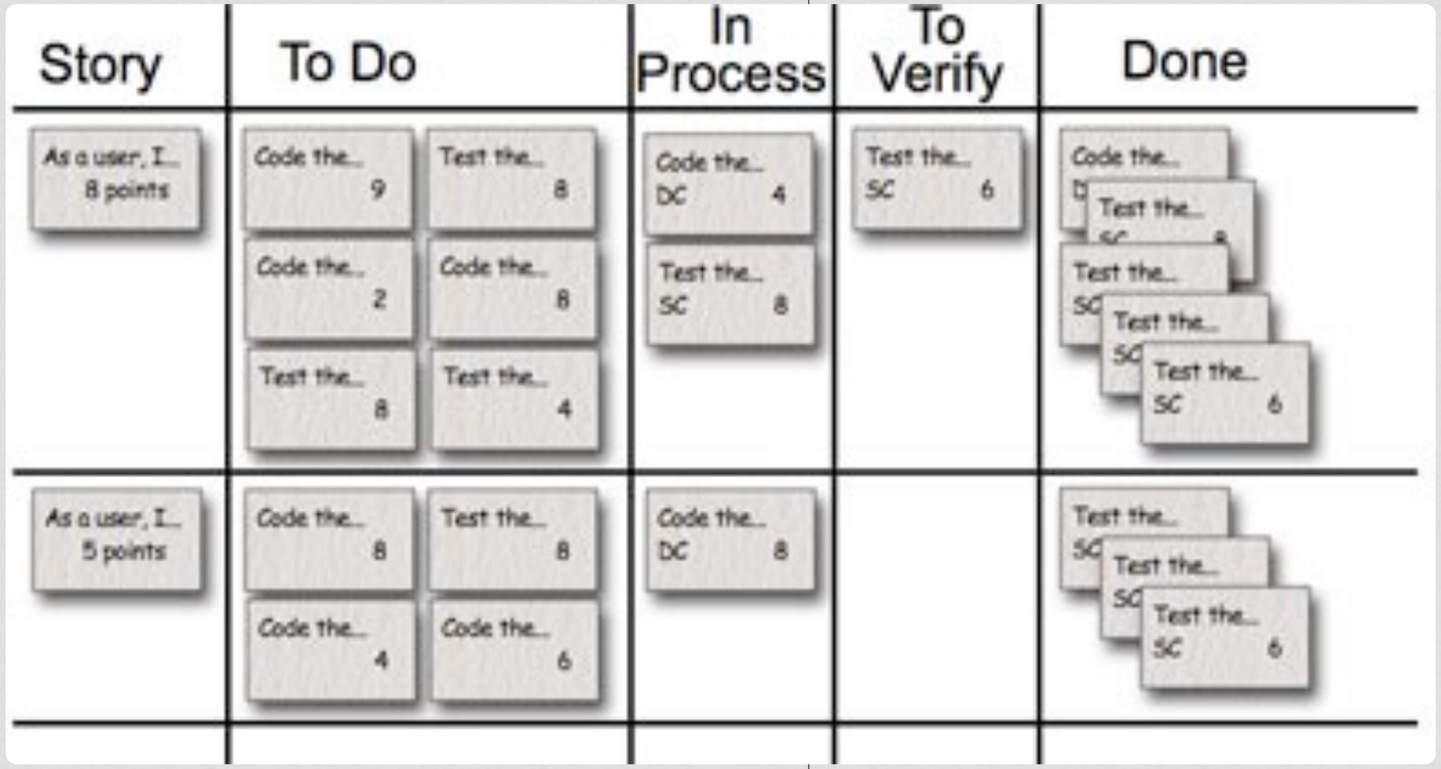
The ScrumMaster's efforts are reflected in the Burndown charts and Impediments Backlog.

The Team

The Team is ideally composed of seven plus or minus 2 members and is cross-functional. There are no designated roles within the team. There are no rules that say a DBA can't work on the web front-end, or even that an artist can't engineer software or vice-versa. This notion can make management uncomfortable. In practice empowering the team to manage itself enables peer pressure to eliminate inappropriate tasking.

The Team is "self-organizing" - yet another notion that makes management uncomfortable. This self-organizing principle promotes accountability, and the transparency inherent in the process reinforces it.

FIGURE 2.2 Scrum Task Board



The Task Board provides provides a quick view of all the current state of the Sprint. It shows the Stories the team is implementing, and the current state of each task in various "swim lanes".

Team responsibilities include:

- Select the iteration goal and specify work results
- Do everything within the boundaries of the project guidelines to reach the iteration goal
- Organize itself and its work
- Demonstrate work results to the Product Owner.

Many Teams have a Scrum Wall somewhere highly visible to all stakeholders. On the wall hang the various artifacts of the process - most importantly the Sprint Backlog and Sprint Burndown chart.

Also frequently used is some form of Task Board (see Figure 2.2). The Task Board is used (by the Team) to manage themselves and illustrates how they manage the work of each Sprint, as illustrated in Figure 2.1 .

Referring again to Figure 2.2, the left column lists all the Product Backlog Items (User Stories in our example), and there is a row for each item. During the Sprint Planning process the story is broken down into the Tasks necessary to manifest the story in working software. On day one of the Sprint all the Tasks are in the To Do column.

Team members simply walk up to the Task Board and assign themselves a task by putting their name on it and moving it to the In Process column. When the task is complete, they move the task to the To Verify column. Once verified (typically by QA), the Task is moved to the Done column.

In this way the work of the Team is 100% visible to anyone who cares to look at the Scrum Wall and Task Board. The Team is self-organizing, and there are a number of positive side effects for engineering best practices like cross-training, code reviews, and the like.

Scrum Ceremonies

There are four ceremonies in Scrum:

- Sprint Planning
- Sprint Review
- Sprint Retrospective
- Daily Scrum

All ceremonies in Scrum are time-boxed and held at regularly scheduled times. Scrum attempts to minimize meetings, so those that exist are deemed important and many Teams impose penalties for tardiness. Times shown are typical for 2 or 4 week Sprints.

Sprint Planning

Sprint Planning (2-4 hrs) is usually composed of two parts. The first part determines what will be done in the Sprint, the second how it will be done. The ScrumMaster facilitates both meetings.

First the Team meets with the Product Owner to review the highest priority items (features) in the Product Backlog, ask questions to clarify understanding, and decide which items will be worked on in the upcoming Sprint. While the discussion is collaborative, the decision about what Product Backlog items will be worked on in the Sprint is the Product Owner's.

In the second part of Sprint Planning, the team breaks down the Product Backlog Items into the tasks necessary to complete the work, and estimates how long each task should take. Note the estimate does not take into account anything about who will be performing the task, as it is not yet known whom that will be. This is another notion that can make management uncomfortable.

Sprint Review

At the end of the Sprint the Team conducts a Sprint Review (2-4 hrs) for the Product Owner, demonstrating in working software all the Product Backlog Items that are 100% "done". Done means the work meets the **Definition of Done** (DoD) as currently defined in the project guidelines.

The Product Owner decides whether to accept or reject each item. If the item is accepted, the item is noted as completed in the Product Backlog, otherwise the item is returned to the Product Backlog where it will be re-prioritized.

Sprint Retrospective

The Sprint Retrospective (1-2 hrs) is held at the end of every Sprint after the Sprint Review meeting. The Team and ScrumMaster meet to discuss what went well and what to improve in the next Sprint. The Product Owner typically does not attend this meeting. Impediments to productivity are added to the Impediment Backlog.

The Sprint Retrospective is an integral part of the inspect and adapt process. Otherwise, the team will never be able to improve their overall output and not focus on the overall team performance.

Daily Scrum

All Team members as well as the ScrumMaster and Product Owner are required to attend the Daily Scrum (15 mins). Anyone else (for example, a departmental VP, a salesperson, or a developer from another project) is allowed to attend but only to listen. This makes the Daily Scrums an excellent way for a Scrum team to disseminate information.

The Daily Scrum is not used as a problem-solving or issue resolution meeting. Issues that are raised are taken offline and usually dealt with by the relevant sub-group immediately after the daily scrum. During the Daily Scrum each team member answers the following three questions:

1. What did you do yesterday?
2. What will you do today?
3. Are there any impediments in your way?

Scrum Artifacts

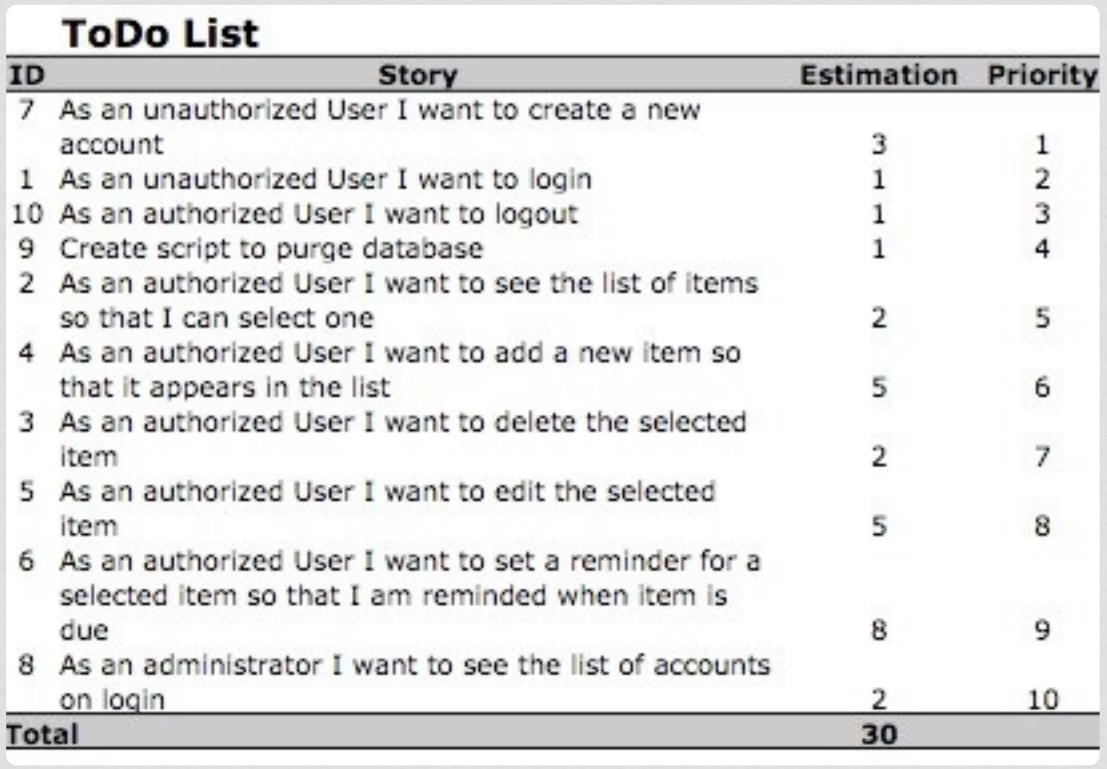
The artifacts of Scrum - like everything else about Scrum and Agile - are designed to be “just enough”. Jeff Sutherland swears by maintaining artifacts on 3x5 cards and colored sticky-notes. Many teams find simple spreadsheets more than adequate tools. There are four artifacts:

- Product Backlog
- Sprint Backlog
- Impediments Backlog
- Burndown Charts

Product Backlog

The Product Backlog is a list of customer requirements prioritized by business value. The Product Backlog includes all features visible to the customer, as well as the technical requirements needed to build the product. The highest priority items in the Product Backlog need to be broken down into small enough chunks to be estimable and testable. Features that will be implemented further out in time can be less detailed.

FIGURE 2.3 Product Backlog



ToDo List

ID	Story	Estimation	Priority
7	As an unauthorized User I want to create a new account	3	1
1	As an unauthorized User I want to login	1	2
10	As an authorized User I want to logout	1	3
9	Create script to purge database	1	4
2	As an authorized User I want to see the list of items so that I can select one	2	5
4	As an authorized User I want to add a new item so that it appears in the list	5	6
3	As an authorized User I want to delete the selected item	2	7
5	As an authorized User I want to edit the selected item	5	8
6	As an authorized User I want to set a reminder for a selected item so that I am reminded when item is due	8	9
8	As an administrator I want to see the list of accounts on login	2	10
Total		30	

The Product Backlog lists all Stories in the project and includes an estimate in relative Story Points.

It is the Product Owner’s responsibility to maintain the Product Backlog. During the current Sprint the Product Owner will schedule a 1-2 hr Backlog Grooming meeting with the Team to refine and clarify the items in the list - especially high priority items that may be considered for the next Sprint. The Product Owner will also meet with business stakeholders, customers, and others to ensure the items are prioritized according to current business needs.

The Scrum Team contributes to the product backlog by estimating the effort of developing features. This effort is expressed in "points" (a.k.a. Story Points), and is a relative measure compared to other features. Note we explicitly avoid estimating effort in units of time. Instead, we estimate the effort in "Story Points" which have no intrinsic meaning. Story Points are relative - e.g., story 3 is about twice as large as story 7.

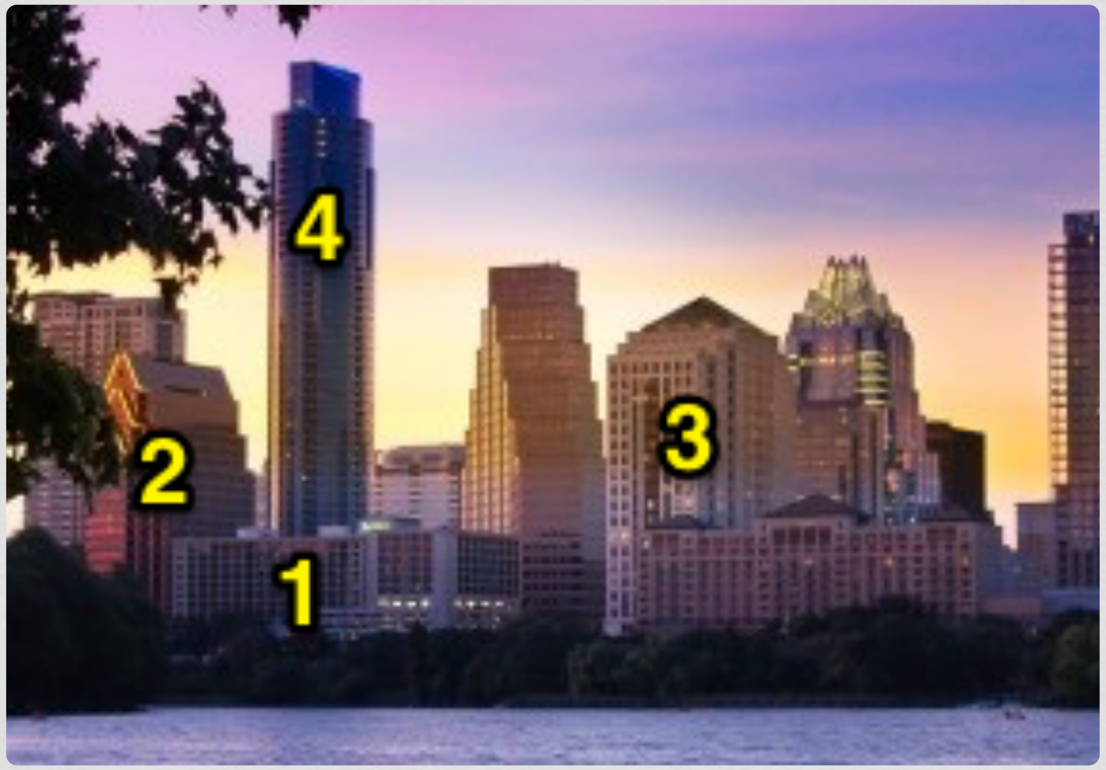
To illustrate relative estimating, let's estimate the height of a few buildings in the Austin Skyline. To estimate the height of building 1 (the Radison), we can count the number of floors, guess that each floor is about 18', and come up with our estimate. Or we can say it's about 1/3 of building 3. Similarly building 2 is about the same as 3, and building 4 is about twice building 3.

Trying to estimate the height in feet is like trying to estimate the entire project in hours. You can do it, but at the beginning of a project your confidence in the estimate is +/-50%.

Most teams use Fibonacci numbers (0, 1, 1, 2, 3, 5, 8, 13, ...) for Story Points and estimate them a Planning Poker session.

Planning Poker is based on the Delphi Method of forecasting and has been shown to be quite reliable.

FIGURE 2.4 Austin Skyline



How tall is each building?

In our Skyline example, we'd probably say building 3 is 5 Story Points, and then calculate the others as 2, 5, 5, and 13 respectively, for a total of 25 Story Points. We then report to management that after carefully estimate the project we have 25 Story

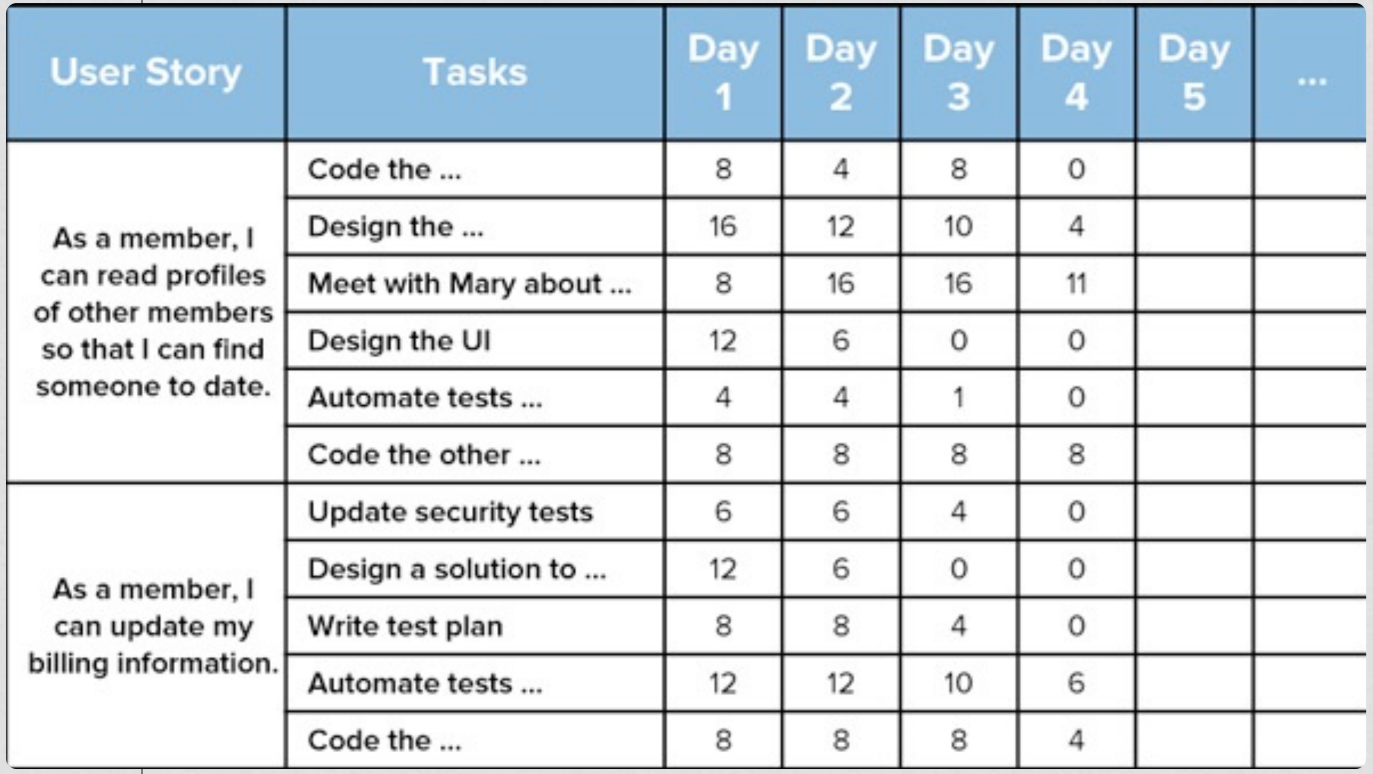
Points! After a moment of silence management asks - fine, but when will it be done?

Scrum answers this question with real, quantifiable data in the form of **Velocity**.

Recall in the Sprint Review the Product Owner accepts or rejects a feature as Done during the Sprint Review. If Done, the item's points contribute to Velocity - "how many Story Points can the Team complete in each fixed duration Sprint". From there, it is trivial to scan the Product Backlog and predict - with surprising accuracy - when any item on the list will be completed.

Returning to our Skyline example - suppose we complete 8 Story Points in the first Sprint. With a Velocity of 8 we can predict it will take 2 more Sprints to complete the remaining 17 Story Points.

FIGURE 2.5 Sprint Backlog



User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	...
As a member, I can read profiles of other members so that I can find someone to date.	Code the ...	8	4	8	0		
	Design the ...	16	12	10	4		
	Meet with Mary about ...	8	16	16	11		
	Design the UI	12	6	0	0		
	Automate tests ...	4	4	1	0		
	Code the other ...	8	8	8	8		
As a member, I can update my billing information.	Update security tests	6	6	4	0		
	Design a solution to ...	12	6	0	0		
	Write test plan	8	8	4	0		
	Automate tests ...	12	12	10	6		
	Code the ...	8	8	8	4		

The Sprint Backlog shows lists the task and remaining hours to complete.

Sprint Backlog

The Sprint Backlog is created when the Scrum Team has committed to deliver a set of top priority features from the Product Backlog. The Product Backlog's features are broken down into a Sprint Backlog - a list of the specific development tasks required to implement a feature. These tasks must be broken down into pieces small enough to accurately estimate - typically a few hours to no more than a few days.

If the estimates indicate the Team is over-committed, the team collaborates with the Product Owner to get the right amount of work into the Sprint to ensure the Team has a high probability to complete the highest priority items successfully during the Sprint.

During a sprint, new tasks may be discovered and tasks already identified may be adjusted. This is perfectly normal behavior. The team simply adds the new tasks (and an estimate of the work remaining on that task) to the sprint backlog or adjusts the wording (and if necessary the estimated work remaining) for tasks in progress. As the team completes tasks, they should be marked on the sprint backlog. The Sprint Backlog shows team members what is complete and what remains. This data will help team members run an effective daily scrum meeting.

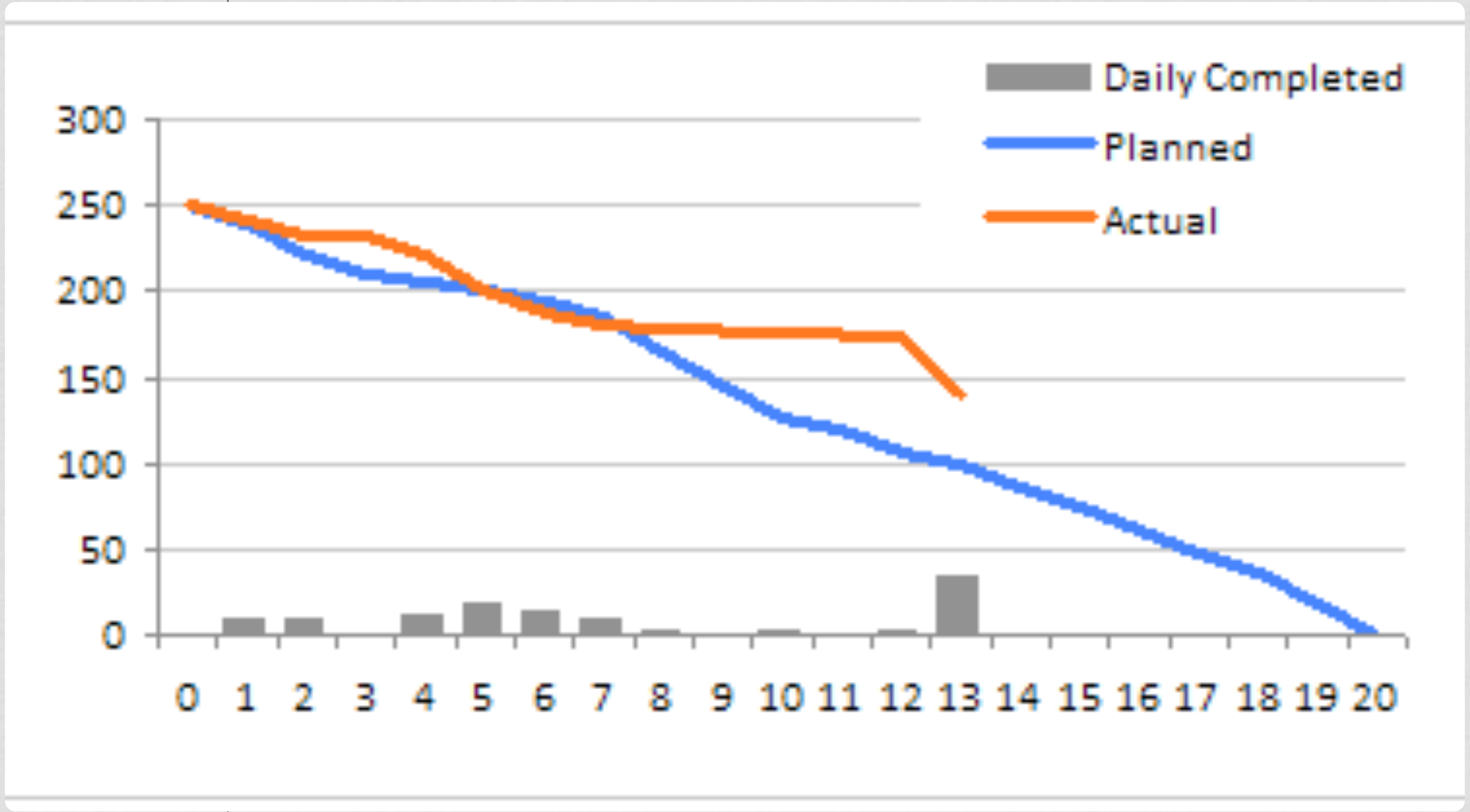
While changes to the sprint backlog are expected, the ScrumMaster should be watching for patterns about the type or amount of work that is added or adjusted. If a pattern emerges, it can teach a team quite a bit about the system as it is built and possibly themselves as a team.

Burndown Charts

The Burndown Charts show the cumulative work remaining in the current Sprint.

At the Sprint Planning Meeting the Team identifies and estimates specific tasks that must be completed for the Sprint to be successful. The total of all Sprint Backlog estimates of work remaining to be completed is the cumulative backlog. When tasks are completed as the Sprint proceeds, the ScrumMaster recalculates the remaining work to be done and the Backlog decreases, or burns down over time.

GALLERY 2.1 Sprint Burndown example.



The Burndown chart depicts the amount of work planned for the Sprint and the amount actually completed so far.

The Product Backlog items brought into the Sprint are fixed for the duration of the Sprint. However, the Sprint Backlog may change for several reasons:

- The Team gains a better understanding of work to be done as time progresses and may find that they need to add new tasks to the Sprint Backlog to complete the Product Backlog items selected. In some cases this could result in an increase in the amount of work planned.

- Defects may be identified and logged as additional tasks. While these are viewed primarily as unfinished work on committed tasks, it may be necessary to keep track of them separately.

- The Product Owner may work with the team during the Sprint to help refine team understanding of the Sprint goal. The ScrumMaster and Team may decide that minor adjustments that do not lengthen the Sprint are appropriate to optimize customer value.

The Burndown Chart is used as a tool to guide the Team to successful completion of a Sprint on time with working code that is potentially shippable as a product.

Agile

Agile software development is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle. The Agile Manifesto introduced the term in 2001.

Related Glossary Terms

Drag related terms here

Index Find Term

Chapter 2 - What is Scrum?

Definition of Done

The exit-criteria to determine whether a product backlog item is complete. In many ways the Definition of Done (DoD) represents the capability of the Team and is the soul of Scrum. A well functioning Scrum Team is constantly improving DoD.

For an immature Team, DoD may be as simple as “passes acceptance test”. As the Team improves, “the code is well commented and has gone through peer review” may be added. Subsequently “automated unit tests written and passed; regression, functionality, integration, and system testing passed; user manual updated; code refactored to extract reusable components”.

Related Glossary Terms

Drag related terms here

Index Find Term

Object Oriented

A type of programming in which programmers define not only the data *type* of a data structure, but also the *methods* (functions) that can be applied to the data structure. In this way, the data structure becomes an object that includes both data and functions. In addition, programmers can create relationships between one object and another. For example, objects can inherit characteristics from other objects.

One of the principal advantages of object-oriented programming techniques over procedural programming techniques is that they enable programmers to create modules that do not need to be changed when a new type of object is added. A programmer can simply create a new object that inherits many of its features from existing objects. This makes object-oriented programs easier to modify.

Related Glossary Terms

Drag related terms here

Index Find Term

Chapter 2 - What is Scrum?

Toyota Production System

The Toyota Production System (TPS) is an integrated socio-technical system, developed by Toyota, that comprises its management philosophy and practices. The TPS organizes manufacturing and logistics for the automobile manufacturer, including interaction with suppliers and customers. The system is a major precursor of the more generic "lean manufacturing." Taiichi Ohno, Shigeo Shingo and Eiji Toyoda developed the system between 1948 and 1975.

Originally called "just-in-time production," it builds on the approach created by the founder of Toyota, Sakichi Toyoda, his son Kiichiro Toyoda, and the engineer Taiichi Ohno. The principles underlying the TPS are embodied in The Toyota Way.

Related Glossary Terms

Drag related terms here

Index Find Term

Chapter 2 - What is Scrum?

Velocity

Velocity is the measure how much product backlog effort a team can handle in one Sprint. This can be estimated by viewing previous Sprints, assuming the team composition and Sprint duration are kept constant.

Once established, Velocity can be used to plan projects and forecast release and product completion dates.

Related Glossary Terms

Drag related terms here

Index Find Term

Chapter 2 - Scrum Artifacts

Waterfall model

The waterfall model is a sequential design process, often used in software development processes, in which progress is seen as flowing steadily downwards (like a waterfall) through the phases of Requirements Definition, Analysis, Design, Implementation, and Testing.

The waterfall development model originates in the manufacturing and construction industries: highly structured physical environments in which after-the-fact changes are prohibitively costly, if not impossible. Since no formal software development methodologies existed at the time, this hardware-oriented model was simply adapted for software development.

Related Glossary Terms

Drag related terms here

Index Find Term