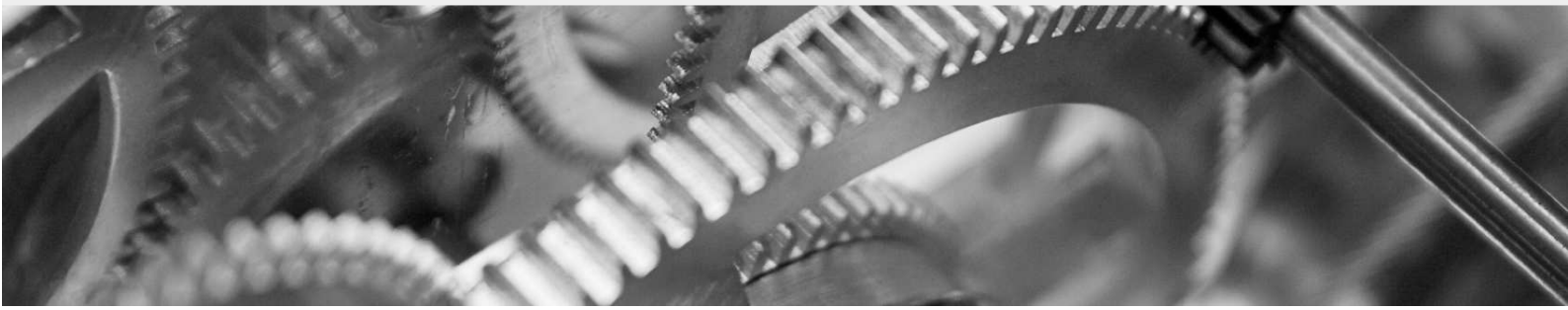
SIGNIFIKANT



SUMMARY OF SCRUM

2007-03-22

STOCKHOLM, SWEDEN: Industrivägen 17, SE-17148 Solna, Sweden. Phone +46 8 735 58 90
MALMÖ, SWEDEN: Jacob Nilsgatan 23B, SE-21121 Malmö, Sweden. Phone +46 40 693 57 66
MUMBAI, INDIA: D/302 Remi Bizcourt, Andheri (West) Mumbai 400 053, India. Phone +91 22 6695 3103

Content

1 Introduction

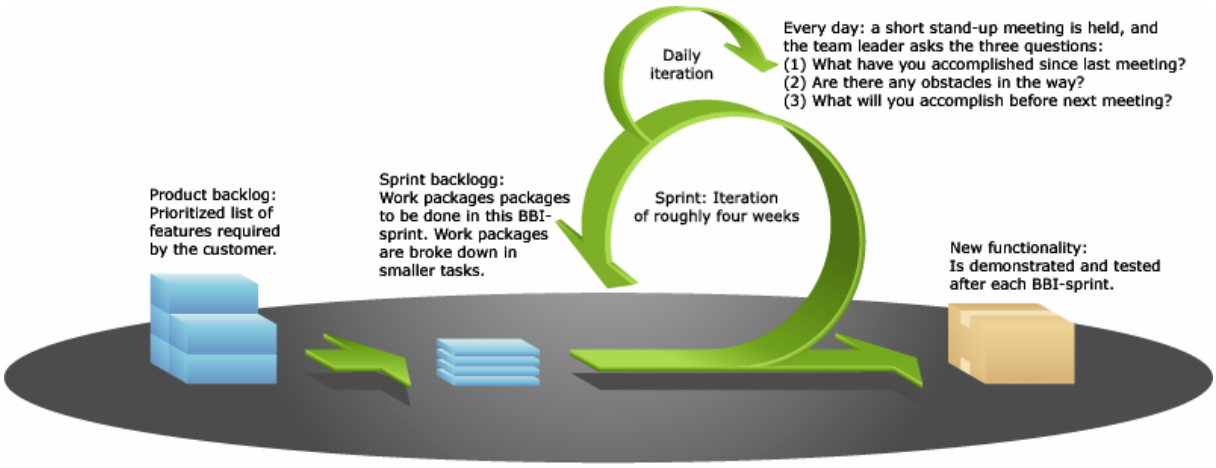
Scrum is an agile process for developing software. With Scrum, projects progress via a series of month-long iterations called sprints.

Scrum is ideally suited for projects with rapidly changing or highly emergent requirements. The work to be done on a Scrum project is listed in the Product Backlog, which is a list of all desired changes to the product. At the start of each Sprint a Sprint Planning Meeting is held during which the Product Owner prioritises the Product Backlog and the Scrum Team selects the tasks they can complete during the coming Sprint. These tasks are then moved from the Product Backlog to the Sprint Backlog.

Each day during the sprint conducts a brief daily meeting called the Daily Scrum, which helps the team stay on track.

At the end of each Sprint the team demonstrates the completed functionality at a Sprint Review Meeting.

Graphically, Scrum looks something like this:



Scrum assumes that the software development process is complicated and unpredictable and treats it as a controlled black box instead of a theoretical, fully defined process. This is one of the biggest differences between Scrum and the Waterfall and Spiral methodologies, which view the software development process as a fully defined process.

Scrum sets a list of loose guidelines that governs the development process of a product, from its design stages to its completion. It aims to cure some common failures of the typical development process, such as:

- **Chaos due to changing requirements** - the real or perceived requirements of a project usually change drastically from the time the product is designed to when it is released. Under most product development methods, all design is done at the beginning of the project, and then no changes are allowed for or made when the requirements change.

- **Unrealistic estimates of time, cost, and quality of the product** - the project management and the developers tend to underestimate how much time and resources a project will take, and how much functionality can be produced within those constraints. In actuality, this usually cannot be accurately predicted at the beginning of the development cycle.
- **Developers are forced to lie about how the project is progressing** - When management underestimates the time and cost needed to reach a certain level of quality, the developers must either lie about how much progress has been made on the product, or face the indignation of the management.

Scrum has been successfully employed by hundreds of different companies in many different fields, with outstanding results.

Scrum is a term from Rugby. Just like in Football we have the line of Scrimmage, in Rugby we have Scrummage == the forwards of each side come together in a tight formation and struggle to gain possession of the ball when it is tossed in among them.

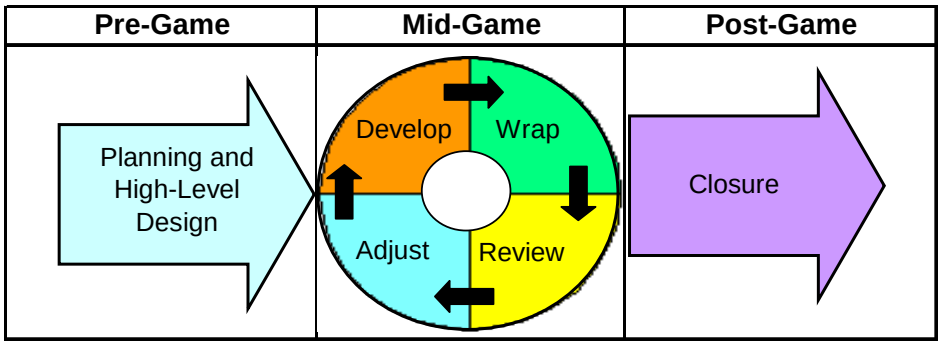
The same concept would apply in this Software approach, where everyone gathers as a team to reach the goal.

2 Scrum Values

The Scrum values are derived from the Agile values of software development.

- **Individuals and interactions over processes and tools** - processes and tools are helpful, but they will do you no good if the team does not communicate and collaborate in a constructive fashion.
- **Working software over comprehensive documentation** - documentation is important, but what's most important is to have working software.
- **Customer collaboration over contract negotiation** - you are not just looking to get a contract and get money that way - you are solving the customer's problem.
- **Responding to change over following a plan** - if the requirements or perceived requirements changed, so should the plans and design.

3 The Scrum Process



The scrum process has 3 main phases.

1.1 Planning

In this phase, the project is planned and high-level design decisions are made.

1.2 Sprint Cycle

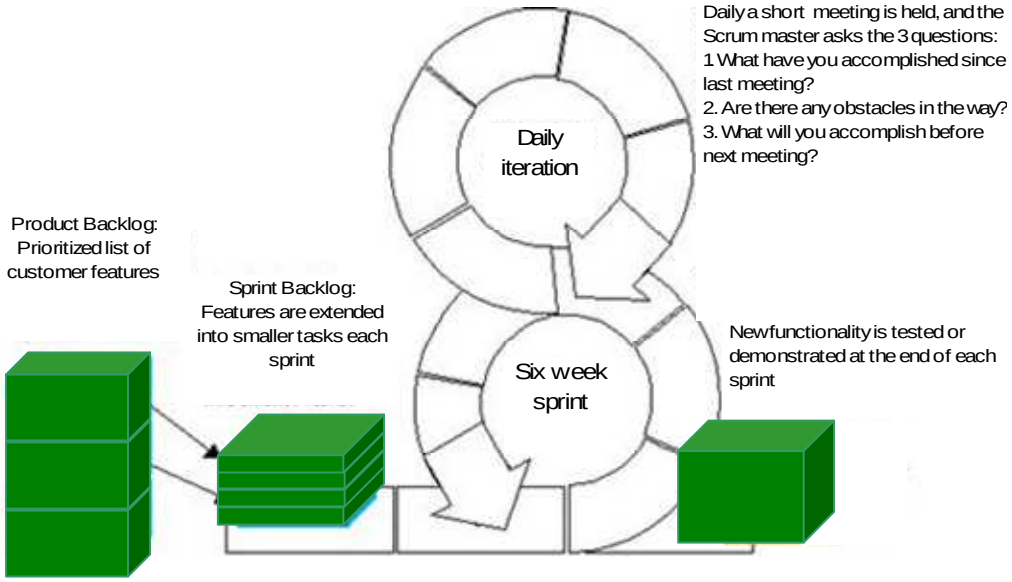


Figure: The SCRUM Sprint Cycle

The sprint cycle is an iterative cycle of about 3-4 weeks, in which the actual development of the product is done. It starts out with a Sprint Planning Meeting to decide what will be done in the current sprint. Then the development is done. A sprint is closed with a Sprint Review Meeting where the progress made in the last sprint is demonstrated, the sprint is reviewed, and adjustments are made to the project as necessary.
The sprint cycle is repeated until the product's development is complete. The product is complete when the variables of time, quality, competition, and cost are at a balance.

- **Develop** the product further - implement, test, and document.
- **Wrap** up the work - get it ready to be evaluated and integrated.
- **Review** the work done in this sprint.
- **Adjust** for any changes in requirements or plans.

1.3 Closure

In this phase, the product's development is brought to a close, and the product is released.

4 Sprint Backlog

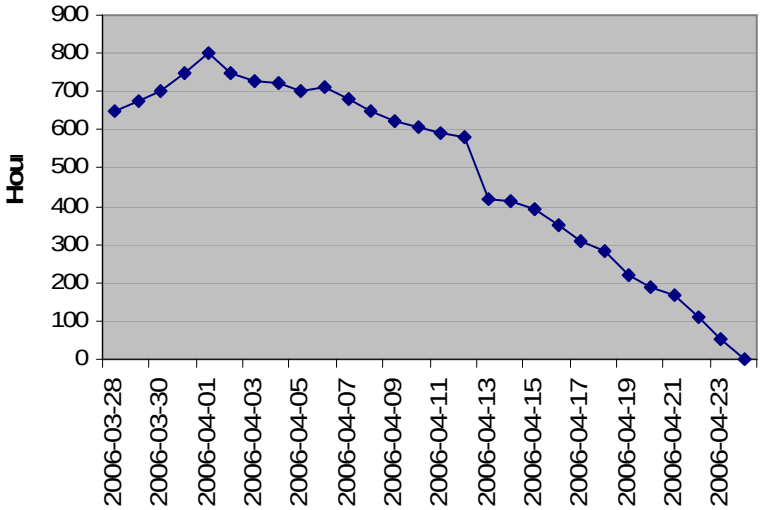
The sprint backlog is the list of tasks that the Scrum team is committing that they will complete in the current sprint. Items on the sprint backlog are drawn from the Product

Backlog, by the team based on the priorities set by the Product Owner and the team's perception of the time it will take to complete the various features.

It is critical that the team selects the items and size of the sprint backlog. Because they are the ones committing to completing the tasks they must be the ones to choose what they are committing to.
The sprint backlog is very commonly maintained as an Excel spreadsheet but it is also possible to use your defect tracking system or any of a number of software products designed specifically for Scrum or agile. An example of the Sprint Backlog in Excel looks like this:

Tasks	Mon	Tues	Wed	Thurs	Fri
Implement algorithm	6	3	6		
Implement XML-parser interface	12	10	8	3	
Implement WebService interfaces	6	15	15	10	6
Add error handling and logging			4	5	5

During the Sprint the ScrumMaster maintains the sprint backlog by updating it to reflect which tasks are completed and how long the team thinks it will take to complete those that are not yet done. The estimated work remaining in the sprint is calculated daily and graphed, resulting in a sprint burndown chart like this one:



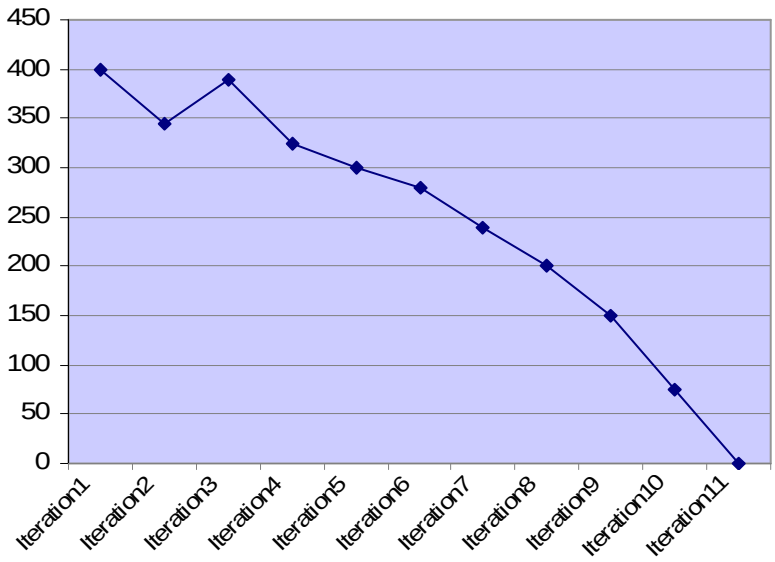
The team does its best to pull the right amount of work into the sprint but sometimes too much or too little work is pulled in during the Sprint planning meeting. In this case the team needs to add or remove tasks. In the above sprint burndown chart you can see that the team had pulled in too much work initially and still had nearly 600 hours to go on 2006-04-11. In this case the Product Owner was consulted and it was agreed to remove some work from the sprint, which resulted in the big drop on the chart between 2006-04-12 and 2006-04-13. From there the team made good consistent progress and finished the sprint successfully.

5 Sprint Burndown

Burn charts have become a favorite way to give visibility into a project's progress. They are extremely simple and astonishingly powerful. They reveal the strategy being used, show the progress made against predictions, and open the door to discussions about how best to proceed, including the difficult discussions about whether to cut scope or extend the schedule. They have a natural mapping to the earned value charts used in military/government projects. They should part of your standard bag of tricks for project planning and reporting. They can be categorized as:

1.1 Release Burndown

On a Scrum project, the team tracks its progress against a release plan by updating a release burndown chart at the end of each sprint. The horizontal axis of the release burndown chart shows the sprints; the vertical axis shows the amount of work remaining at the start of each sprint. Work remaining can be shown in whatever unit the team prefers--story points, ideal days, team days, and so on. My preference is for story points, for all of the reasons described in the Agile Estimating and Planning book.

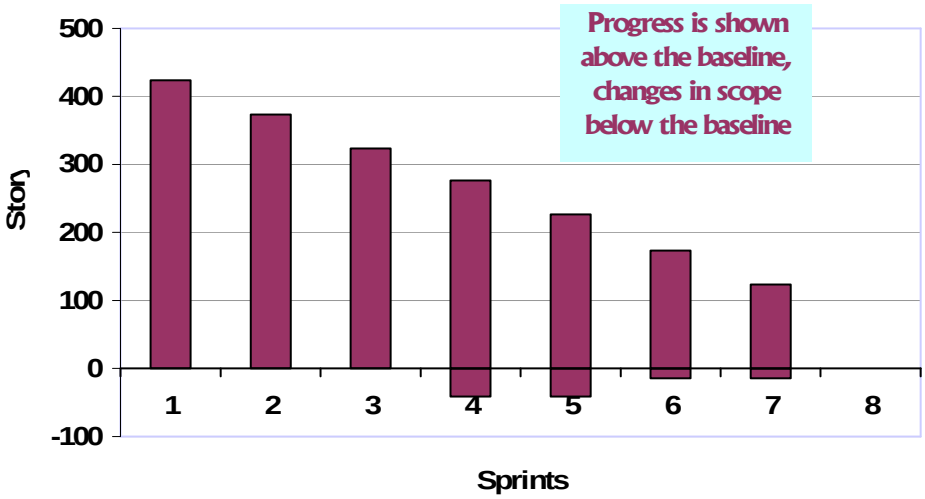


On this burndown chart, the team started a project that was planned to be eleven two-week sprints. They began with 200 story points of work. The first sprint went well and from the chart you can infer that they had around 180 story points of work remaining after the first sprint. During the second sprint, however, the estimated work remaining actually burned up. This could have been because work was added to the project or because the team changed some estimates of the remaining work. From there the project continued well. Progress slowed during sprint 7 but then quickly resumed.

This type of release burndown chart works very well in many situations and in many teams. However, on projects with lots of changing requirements you may want to look at an alternative release burndown chart.

1.2 An Alternative Release Burndown Chart

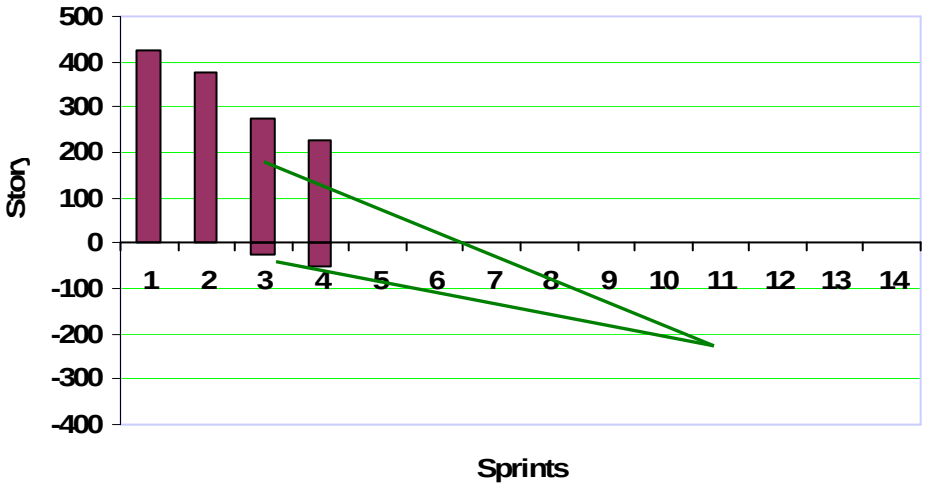
The typical Scrum release burndown chart shows a single value--the net change in the amount of work remaining. In some cases the simplicity of this is wonderful. However, it can also mask what may be going on in a project. For example, suppose a team had expected to make progress of 40 (hours, points, whatever) last sprint but the burndown chart only shows net progress of 10. Was the team slower than expected or was more work added to the release? It's important to know the answer to this question because we cannot really predict when the release will be done without it. With this in mind, I've introduced the following type of burndown chart:



On this burndown chart, the height of each bar represents the amount of work remaining in the release. I work prefer to estimate product backlog, items in "story points" so this figure shows a release with 175 story points planned in it as of sprint 1. The team finished 25 points in sprint 1, leaving 150 to go as of the start of sprint 2. There were 120 as of the start of sprint 3. So, the top of the bar is reduced by the amount of work the team finishes in a given sprint. Before the start of sprint 4, the product owner added work to the project. This additional work is shown at the bottom of the bar for the fourth sprint. You can see that the vertical height of sprint 4 goes from about -40 to about 95, or 135 points of work remaining. Fourty of those 135 points are from new work.

Prior to the start of sprint 6 work was removed by the product owner. As with an increase in scope, a decrease in scope comes off the bottom. This is true whether the work removed is work that was initially planned or work that was added during the project.
One way to predict how many sprints a project will take is to draw a trend line through the bars and extend the baseline. For example:

A problem with this is that predicting the end date as above does not include the rate of change to the scope of the project. You can anticipate the number of sprints needed by also drawing a trend line through the changes occurring at the bottom of the bars as shown below:



1.3 Velocity

Velocity is a technique very useful during sprint planning. Take all backlog items that were scheduled for last sprint (i.e. initially included in the sprint backlog for last sprint) and sum up their initial time estimates (story points) or just look at the first Y-value ("estimated remaining work") on the burndown chart from last sprint. This is your *estimated* velocity for last sprint.

For the actual velocity look at which backlog items were actually completed (ignore partially completed items). Sum up the *initial estimates* for those items (ignore any time estimate updates that were done during the sprint). This is your *actual* velocity.

Last sprint		This sprint	
Estimated velocity	65	Total mandays:	80

Actual velocity:	44	Focus:	75%
		Estimated velocity:	60

Figure - velocity of a sprint

It is a very simple measurement, and very rough, but it shows one hard fact. “Regardless of the reasons, here is the approximate difference between how much we thought we would get done and how much we actually got done”.

The basic formula for ”Total man-days” is **teamsize * sprintdays**.

If the team size is 7 people and the sprint length is 3 weeks, that’s 7 people * 3 weeks * 5 sprint days per week, or 105. Subtract part timers, vacations and other time that may disappear. Let’s say this ends up being 80 man-days.

The *focus factor* is used to scale down the available man-days to a more realistic estimated velocity. We call this the *focus factor*. The focus factor number is usually around 50 – 80 %. The value will get more accurate as you gain experience in sprint planning. In the example above, the focus factor is used to translate “total man-days” (80) into “estimated velocity” (60 story points).

Estimated velocity for this sprint is directly useful for two reasons:

1. Estimated velocity acts as a natural limit to how much work the team should commit to during this sprint.
2. The estimated velocity can (and should) be compared to the last sprint.

6 Scrum whiteboard

The process is very simple but it’s really incredibly effective. Buy a whiteboard. For each person in the sprint, add a row on the whiteboard (preferably using tape designed to make squares on whiteboards). Above each person's name add 3 columns: not started, in progress, complete. A larger whiteboard can actually hold two full name/status sections on the board. Each square should be around 6x6, maybe 6x8. If you can't fit the whole team on a large whiteboard, you probably have other issues with scrum.

At the beginning of each sprint, people commit to tasks and write a summary of the task on a specific color of post-it-note, the larger 4x4 post-it notes work best. These will be the committed tasks for the sprint. They also write the estimated hours on the note. You also need at least one other color of post-it notes, and those are for add-on or unplanned tasks. Those should also have estimates depending on the situation. For a while we also had a 3rd color of post-it notes to track specific bugs, but I didn't see as much benefit in that. No matter how many colors of notes you have, they should all try to follow the not-started->in-progress->complete left-to-right migration.

At the beginning of the sprint, all the post-it notes should be in the not started column. As people start tasks, they move them to the in progress column. You can have as many tasks as you want in the in progress column. Or none if you're a slacker. Every day the number of hours spent on each task in progress should be updated and written onto the note. When a task is finished, the person should sum up the hours on the post-it note and write the total on time then circle it to indicate it's the final. You can use the back of the post-it if needed. If it

was a small task and it only one number on it because the estimate and actual match, then just circle that number, so long as you're honest and nice to your PM and circle the actuals.

Now you have a view of the estimates vs. actuals for every task. You can quickly and easily see if unplanned tasks are killing your sprint. If resources are imbalanced in terms of hours or numbers of tasks that should quickly be visible as well.

If tasks don't get finished, they roll over easily under this model. All you do is move them back to not started for the next sprint and previous sprint's work history for the task is retained on the post-it note.

You need someone in a leadership role who takes the responsibility of tracking this information, preferably translating some of it like actuals/estimates and unplanned items back into digital form. The overhead of doing this was comparable to every other method I've seen used.

This was the most efficient way to have people report their status and actuals without feeling put out by it. No one on the team seemed mind the format, and it was almost fun to keep moving things (hopefully) from left to right throughout the sprint. Its like giving the entire team a quick and very visual view of the progress during every daily scrum. Plus, if you have people on your team who aren't all that good at articulating what they did verbally, it gives them an opportunity to show their good progress and still be a little quiet about it.

7 The Scrum Team

The SCRUM team consists of 2 groups - the interested team, which consists of people who are interested, but who will not be doing the work, and the working team - people who are interested, and will be doing the work on the project.

A team typically has no more than 6-9 working members, although SCRUM has been successfully used with more members. If there are more members than manageable, the project should be broken into multiple sub-projects, each focusing on one, self-contained area of work (one for QA, one for documentation, etc.). There should be people to act as bridges - that is, to attend the meetings of more than one SCRUM team, and act as a communication bridge between the teams. Members of teams that are closely related/involved with each other should sit in on the other teams' SCRUM meetings.

i. The leader (Scrum Master)

The team's leader is called the Scrum Master. He should be one of the members of the working team - that is, he should be one of the people who is actually doing the work on the project. The Scrum Master coaches the development team, removes any possible impediments and constantly works to ensure that the team has the best possible circumstances for realizing the goals fixed for the Sprint. Each Sprint enhances the product's market value and adds new functions and improvements that can be delivered to the customer.

8 Challenges

One of the main challenges for companies adopting Scrum is how to deal with the huge cultural change involved in moving to an approach based on small teams, and the costs

associated with that. "It's not cheap, you've got to change and retool the culture," says Mr Dorsey at WildCard. "The training costs are significant."

On top of this, companies need "a lot of really good leadership skills," according to Bob Schatz, chief development officer at Solstice Software, which makes software for enterprises to test their integration infrastructure. "Many people have a lot invested in the old way of doing things, and it becomes more about ego, but you have to drop that."

Some software developers, too, may prefer to work in cubicles, receiving orders from above. For these more introverted types, says Mr Schwaber, it helps to have an approach such as Scrum, with its clear rules for how people should work together, much as in football or rugby.

9 Certification

i. **ScrumMaster Certification Course** – for mastering and coaching the development team, removes any possible impediments

ii. **Scrum Methodology** - for implementing Scrum throughout an organization and for meeting CMM Level 3 requirements.

iii. **Individual Licenses of Scrum** - If you are a professional who intends to teach others how to benefit from Scrum, get certified and receive your own copy of Scrum.

iv. **Organizational Licenses for Scrum** - Do you want to use Scrum to manage one or more of your projects? Get an organizational license.

10 References

i. Booch, Grady. Object Oriented Analysis and Design with Applications. The Benjamin/Cummings Publishing Company, Inc., 1994, p. 8

ii. Booch, Grady. Object Solutions: Managing the Object-Oriented Project. Addison-Wesley, 1995.

iii. Brooks, F.P. “No silver bullet—essence and accidents of software engineering.” Computer 20:4:10-19, April 1987.

iv. Coplien, J. “Borland Software Craftsmanship: A New Look at Process, Quality and Productivity.” Proceedings of the 5th Annual Borland International Conference, June 5, 1994. Orlando, Florida.

v. DeGrace, P. and Hulet Stahl, L. 1990. Wicked Problems, Righteous Solutions. Yourdon Press

vi. Ogunnaike, B. 1994. Process Dynamics, Modeling, and Control. Oxford University Press.

vii. Pittman, Matthew. Lessons Learned in Managing Object-Oriented Development. IEEE Software, January, 1993, pp. 43-53.

viii. Rumbaugh, October 1995, “What Is a Method”. Journal of Object Oriented Programming. Schwaber, Ken. “Controlled Chaos: Living on the Edge.” American Programmer,April 1996.

ix. Sutherland, Jeff. ScrumWeb Home Page: A Guide to the SCRUM Development Process. Jeff Sutherland’s Object Technology Web Page, 1996 http://www.tiac.net/users/jsuth/scrum/index.html

x. Takeuchi, Hirotaka and Nonaka, Ikujiro. January-February 1986. “The New New Product Development Game.” Harvard Business Review.