Teams that Finish Early Accelerate Faster: A Pattern Language for High Performing Scrum Teams

Jeff Sutherland
Scrum Inc.
jeff@scruminc.com

Neil Harrison
Utah Valley University
neil.harrison@uvu.edu

Joel Riddle
Scrum Inc.
joel@scruminc.com

Abstract

Recent surveys show that only 42% of Agile projects are successful. While this is three times better than traditional projects, 49% of Agile projects are late or over budget and 9% are total failures [1]. Is there a way to help Agile teams to execute better? At the 2013 Scrum PLoP Conference held in Tisvildeleje, Denmark, thought leaders in the Agile community reviewed a set of Scrum Patterns that together generate a high performing Scrum team. During this editorial process it became apparent that a combination of nine Patterns in conjunction with the Scrum framework could help teams achieve hyperproductivity while simultaneously increasing revenue per point. Hyperproductivity is defined as a 400% increase in velocity over a teams initial velocity.

1. Introduction

Many years before the writing of the Agile Manifesto [2], Mike Beedle was influenced by the online description of Scrum [3], implemented the process in his own company, and led the effort to drive Scrum through the Pattern Languages of Programming Design conferences. The result was **Scrum: A Pattern Language for Hyperproductive Software Development**, the first (and only) published organizational pattern that describes a complete Agile process [4].

Recent work by Jim Coplien shows that Scrum is deceptively simply while compressing a complex array of organizational patterns [5]. Jim was surprised when he found that Scrum incorporates at least 33 organizational patterns into a framework that can be explained in 2 minutes.

One of Scrum's design goals was to encapsulate best practices from 40 years of software development into a process that was simple enough for the average developer to use with less than 2 days of startup time. Jim's research shows that we did a good job of accomplishing that goal.

In recent years the Scrum Pattern Community has evolved a comprehensive set of patterns for Scrum [6] that allow teams to try proven approaches that have worked in many companies. While the Scrum Guide [7] provides the basic rules of Scrum, the patterns amplify the guide by showing teams how to solve problems in a specific context.

2. Hyperproductive Software Development

While Scrum is a pattern language for hyperproductive development, only a small percentage of Scrum teams have achieve Scrum's design goal of 5-10 times traditional project productivity with a corresponding increase in quality. In addition to Mike Beedle's [3] and Jeff Sutherland's companies [8], examples are seen in the U.S. [9], Russia [10], the Netherlands and India [11], and from Software Productivity Research data on agile teams [12].

Systematic, a CMMI Level 5 company in Denmark, has shown how to systematically produce a hyperproductive team by focusing on a high standard for "Done" at the end of a sprint and "Ready" at the beginning of a sprint [13]. They noticed that it was impossible to achieve hyperproductivity if they changed members of the Scrum team at the beginning of every project, showing that the pattern **Stable Teams** is a requirement for high performance. Similar results were observed consistently for a style of Scrum called "Shock Therapy" in the U.S. and Europe [14].

These two approaches to consistently generating a hyperproductive team have been either too disciplined or too aggressive for most Scrum teams to implement. However, a venture capital group with over 30 companies doing Scrum decided to implement Scrum internally in 2006 for all departments in the venture company [15]. After running hundreds of sprints with good metrics and analyzing the data, they discovered that **Teams that Finish Early Accelerate Faster** [16]. This provided

a clue for the average team. If a stable team could accelerate faster by finishing early, what other simple steps could be taken by any team to achieve **Scrum: A Pattern Language for Hyperproductive Software Development**?

3. The Secret Sauce: A Generative Pattern Language

A Pattern Language is an attempt to express the deeper wisdom of what brings aliveness within a particular field of human endeavor, through a set of interconnected expressions arising from that wisdom… It's more than a set of tools in a toolbox. It moves beyond a list of processes, to seek activities or qualities that repeat across many of those processes … in an effort to home in on what works. It is an interconnected whole, that when applied coherently, brings "the quality that has no name" (QWAN) into a field of human endeavor [17].

The investors at OpenView Venture Partners were surprised when they discovered **Teams that Finish Early Accelerate Faster**. They observed that Scrum is not about velocity, it is about acceleration! An accelerating team will soon outperform any good team whose velocity flatlines.

This pattern seemed counterintuitive to the investors, so the authors and others experimented with it in other companies and found that it consistently worked. The next question become how to get it to work well enough to generate a hyperproductive team. What set of *generative* patterns will feed off one another, generating unexpected side effects that keep teams accelerating.

Generative patterns work indirectly; they work on the underlying structure of a problem (which may not be manifest in the problem) rather than attacking the problem directly. Good design patterns are like that: they encode the deep structure of a solution and its associated forces, rather than cataloging a solution [18].

We already knew from the Systematic data [13] that **Stable Teams** were necessary for hyperproductivity. We decided to systematically investigate every other major problem that blocks a team from finishing early.

4. The Patterns

A Scrum Pattern is an auxiliary process that helps solve known problems when implementing the Scrum framework. The structure of Scrum is simple and designed to help Teams adapt to change as it occurs but Scrum doesn't solve every problem. As Scrum has been implemented and improved upon over time, a number of practices evolved to address common problems. These are Scrum Patterns.

Every year at the Scrum PLoP conference, new Patterns are proposed and go through a round robin editorial process by some of the most influential minds in the Scrum community. Eventually, if the Pattern is seen as having value, it is approved and added to the Pattern spreadsheet.

As more and more Patterns emerge, they can be used together. The nine Patterns listed below are in essence the vocabulary of the *Pattern Language for Hyperproductive Teams*.

The Patterns are:

1. Stable Teams
2. Yesterday's Weather
3. Swarming: One Piece Continuous Flow
4. Interrupt Pattern: Illigitimus Non Interruptus
5. Daily Clean Code
6. Emergency Procedure
7. Scrumming the Scrum
8. Happiness Metric
9. Teams that Finish Early Accelerate Faster

The first two patterns help the team get ready for a successful sprint. Patterns 3-6 help the team deal with the most common disruptive problems in a sprint. Patterns 7-8 will drive a team to the hyperproductive state by causing Pattern 9 to emerge as a side effect.

5. Patterns that Help Teams Get Ready

Stable Teams: *Keep teams stable and avoid shuffling people between teams. Stable teams tend to get to know their capacity, which makes it possible for the business have some predictability.*

The Scrum framework is built around a Team of three to nine members. Small Teams keep communication paths simple and allow for communication saturation. However, just having a small Team doesn't mean it will be successful. If members are often pulled off the team to work on other projects or are unable to participate regularly in rituals, the team's Velocity will suffer. To solve this issue practitioners realized they not only needed small teams but stable teams.

At PatientKeeper [19] during 2005-2007 all teams were hyperproductive except an offshore waterfall team. Careful data collection during this period showed the onshore teams were 10 times as productive as the offshore team. A key feature was the stability of the onshore teams with almost no changes in team members during this period. We discovered a new person added to the team about

every 6-12 months was helpful to bring in fresh ideas.

Stable teams reach a consistent Velocity, ideally increasing about 10% a Sprint until they achieve Hyper-Productivity. A consistent Velocity helps the Team predict how many Points they can accomplish each Sprint.

Yesterday's Weather: *In most cases, the number of Estimation Points completed in the last Sprint is the most reliable predictor of how many Estimation Points will be completed in the next Sprint.*

Yesterday's Weather allows teams to build a more accurate Sprint Log limiting the possibility of the team ambitiously pulling in too many Estimation Points and endangering the Sprint. *Stable Teams know their capacity, which enables them to use Yesterday's Weather*.

6. Patterns that Help Teams Finish the Sprint

Once stable teams have built a realistic Sprint Log using *Yesterday's Weather*, they start their Sprint. There are numerous forces that can cause a Sprint to fail. The following four Patterns are designed to address the most common Sprint pitfalls.

Swarming: *Focuses maximum team effort on one item in the Sprint Backlog to get it done as soon as possible. Whoever takes this item is Captain of the team. Everyone must help the Captain if they can and no one can interrupt the Captain. As soon as the Captain is Done, whoever takes responsibility for the next priority backlog item is the new Captain.*

When Team struggle to finish Sprints, it is usually because they have to much *Work in Process* and aren't focusing on getting the highest business value Product Backlog Item finished. Swarming helps teams move items to "Done" quickly, increasing Velocity. *Yesterday's Weather allows Swarming Teams to increase Velocity because the team is building a realistic Sprint Log.*

Interrupt Pattern: *Allot time for interruptions and do not allow the time to be exceeded. Set up three simple rules that will cause the company to self-organize to avoid disrupting production*:

1. The team creates a buffer for unexpected items based on historical data. For example, 30% of the team's work on the average is caused by unplanned work coming into the sprint unexpectedly. If the team velocity averages 60 points, 20 points will be reserved for the interrupt buffer.
2. All requests must go through the Product Owner for triage. The Product Owner will give some items low priority if there is no perceived value relative to the business plan. Many other items will be pushed to subsequent Sprints even if they have immediate value. A few items are critical and must be done in the current Sprint, so the Product Owner puts them into the interrupt buffer.
3. If the buffer starts to overflow, i.e. the Product Owner puts one point more than 20 points into the Sprint, the team must automatically abort, the Sprint must be re-planned, and management is notified that delivery dates will slip.

The Interrupt Pattern, like Swarming, allows teams to finish their Sprints because they have developed a process to deal with found work.

Balihoo, and OpenView Venture Partners portfolio company failed 18 two-week sprints in a row. After implementing this pattern, all sprints were successful, none were aborted, and velocity more than tripled. The pattern generates a side effect that causes the entire company to self-organize to avoid sprint aborts. This means the buffer is never completed used up and teams tend to finish early and put forward from the next sprints backlog.

Daily Clean Code: *Fix all bugs in less than a day. Aim to have a completely clean base of code at the end of every day.*

If a Team isn't creating daily clean code, a lot of time can be wasted going back to fix bugs. Errors can be limited by building quality control into the development process so that issues are discovered and corrected at the point of origin. Research in Silicon Valley at Palm, Inc. in 2006, showed that a bug that isn't fixed the day it's created can take as much as 24 times longer to correct three weeks later.

By creating clean code daily, the Team will limit interruptions later in the Sprint or in the next Sprint when the error is harder to correct. The idea is that once an error has been caught, analyzed and corrected, it should never happen again. By eliminating errors and improving the process on a daily basis, teams can continuously improve, increasing quality with corresponding Velocity.

Emergency Procedure: *When high on the burndown try a technique used routinely by pilots.*

When bad things happen, execute the emergency procedure designed specifically for the problem. Do not delay execution while trying to figure out what is wrong or what to do. In a fighter aircraft you could be dead in less time than it takes to figure out what is going on. It is the responsibility of the Scrum Master to make sure the team immediately executes the Scrum Emergency Procedure, preferably by mid-sprint, when things are going off track.

Emergency Procedure Steps: (do only as much as necessary)

1. Change the way the work is done. Do something different.
2. Get help, usually by offloading backlog to someone else.
3. Reduce scope
4. Abort the sprint and replan
5. Inform management how release dates will be affected

7. Getting Hyperproductive

Stable Teams and *Yesterday's Weather* set the team up for success by helping it get in a ready state. *Swarming, the Interrupt Pattern, Daily Clean Code* and *Stop the Line* help the Team deal with Impediments as the arise during the Sprint. The last three Patterns take advantage of these previous Patterns and allow the team to attain a hyperproductive state.

Scrumming the Scrum: *Identify the single most important impediment in the Sprint Retrospective and remove it before the end of the next sprint. To remove the top impediment, put it in the Sprint Backlog as a user story with acceptance tests that will determine when it is Done. Then evaluate the state of the story in the Sprint Review like any other task.*

If the team is able to capitalize on *Scrumming the Scrum* they should create at least one process improvement per sprint. This contributes to increasing Velocity 10% every Sprint. If the team is using *Yesterday's Weather*, than they have a good chance to finish their sprint early because they will have one less impediment dragging down their Velocity.

Happiness Metric: *Happiness is one of the best metrics because it is a predictive indicator. When people think about how happy they are they are really projecting out into the future about how they feel. If they feel the company is in trouble or doing the wrong thing, they will be unhappy. Or if there is a major roadblock or frustrating system they have to deal with, they will be unhappy.*

A powerful way to take the pulse of the Team is by finding out how happy they are. The Scrum Master asks just 2 questions:

- How happy are you with the company?
- How happy are you with your role?

Team Members are asked to rate their feelings on these questions on a scale from one to five. These numbers are kept in a spreadsheet and tracked over weeks. Whenever the average changes significantly it's important to talk about it and see how a Team can be made happier. By monitoring the team's happiness, the Scrum Master can anticipate drops in Velocity and make adjustments.

Teams That Finish Early, Accelerate Faster: *Teams often take too much work into a sprint and cannot finish it. Failure prevents the team from improving. Therefore, take less work into a sprint. Maximize your probability of success by using the pattern* ***Yesterday's Weather****. Then implement the four Patterns that reduce Impediments within the Sprint, which will systematically deal with any interruptions and help you finish early. On early completion pull forward from the next Product backlog which will increase Yesterday's Weather for future sprints. To increase the probability of acceleration, apply* ***Scrumming the Scrum*** *to identify your kaizen in the retrospective. Put the kaizen in the sprint backlog with acceptance tests for the next sprint as top priority.*

8. Example

A team used the Happiness Metric as a way to identify and prioritize process improvements. On a scale of 1-5 they ask (1) how do they feel about their role in the company and (2) how do they feel about the company. Then they shared what would make them feel better. The team used planning poker to estimate the value of things that would make team members feel better. The team estimated the value (as opposed to effort) of backlog items as well. The entire product backlog was estimated at 50 points of value.

"Better user stories" was the top priority improvement for the team. Removing this impediment was estimated at over 60 points of value. The Chief Product Owner wondered if removing that impediment might double velocity, as the

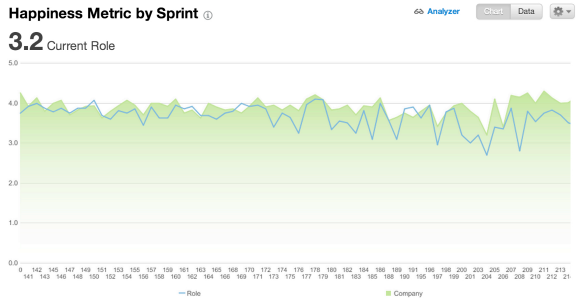
impediment value was higher than the entire product backlog value for the sprint.

"Improve User Stories" was put into the Product Backlog and pulled into the next sprint with a definition of Done. This definition of Done was implemented as acceptance tests that had metrics that were calculated at the next sprint review. They included:

1. How many stories got into the sprint that did not meet the INVEST criteria (immediately actionable, negotiable, valuable, estimable, sized to fit, and testable)?
2. How many times did developers have to go back to the product owner to clarify a story during a sprint?
3. How many times did dependencies force a story into a hold state during a sprint?
4. How many stories had process efficiency of over 50%? (process efficiency = actual work time/calendar time)
5. How many stories were not clear to developers? Measure by number of teams members that complained about a story.
6. How many stories implied technical implementation rather than clarifying desired user experience?
7. For how many stories did developers understand the linkage between the story, the theme that produced the story, the epic that generated the theme, and the business need that generated the epic? Measured by number of team members complaining that they did not understand why they were doing a story.

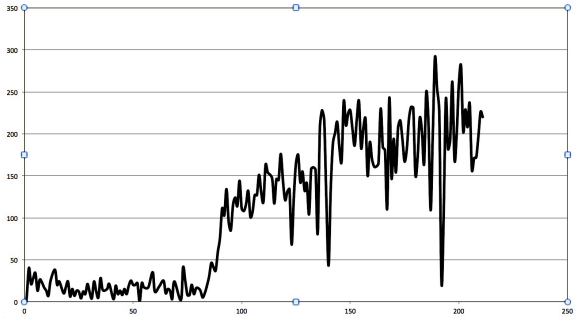
Resulting Context:

While improving the quality of user stories is never ending, the sprint review demonstrated significant improvement on this backlog item as measured by the acceptance tests. Significant improvement resulted in an increase in velocity sprint to sprint for three sprints. After velocity had tripled this impediment fell off the top of the impediment list and another impediment took its place.



The graph above is team happiness data for weekly sprints 140-212 where the dark green line is happiness about the individual's work and the light green is happiness about the company.

The graph below shows the raw velocity of the team. In Sprint 86 the team was doubled in size. By sprint 89, "Improve User Stories" was put in the backlog of each sprint. Within three sprints velocity tripled. By Sprint 211 output was up 1200% and the team had tripled in size. This is the first documented, sustainable, hyperproductive company, as the data include all work for the entire company. The low points on the velocity graph are where the company was on holiday.

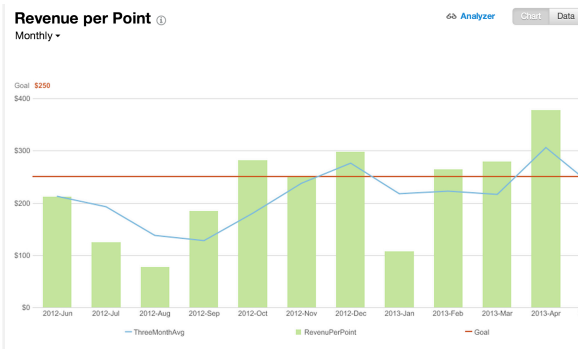


Source: Scrum Inc. Company Data 2010-2013, weekly sprints 1-214

9. Caveats

As the example team accelerated, they were concerned about point inflation and the management was concerned about value delivery. There were several factors that resolved this problem.

1. Point deflation is the most common issue for teams. As work becomes easier they estimate less points. This flatlines velocity.
2. The team meets regularly to review a set of reference stories to assure points are stable.
3. The management and product owner encouraged the team to break all stories down to 1, 2, or 3 points. If all stories are small you can just count stories to double check acceleration.
4. The most important factor, however, is for management to measure product owner performance by challenging them to double revenue per point.



For this team, the revenue per point was $212 in June of 2012. By April of 2013 revenue per point was $377. A great Scrum team with a great Product Owner can quadruple the number of points while doubling the revenue per point, achieving 800% revenue growth. This makes the management very happy.

10. Conclusions

By implementing and executing all nine Patterns, teams dramatically increase their ability to finish the Sprint early. This allows them to pull more Product Backlog Items from the Product Backlog. This will increase Velocity and establish a higher baseline for *Yesterday's Weather*, setting the team-up for the next Sprint. Teams that finish early also tend to have a higher Happiness Metric because they feel confident about their ability to complete Sprints. This in-turn starts a virtuous cycle of continuous improvement eventually leading to Hyper-Productivity.

The generative nature of these patterns is not obvious to those who have not tried them. Unanticipated side effects cause unexpected positive results. Therefore, it is recommended that all teams try these patterns, particularly in combination, to see if they help improve performance, quality, and happiness of the team.

11. References

[1] K. Schwaber and J. V. Sutherland, *Software in 30 days : how agile managers beat the odds, delight their customers, and leave competitors in the dust*. Hoboken, N.J.: John Wiley & Sons, Inc., 2012.

[2] M. Fowler and J. Highsmith, "The Agile Manifesto," *Dr. Dobbs,* July 13 2001.

[3] M. Beedle. (2010, June 15). *Mike Beedle on the Early History of Scrum*. Available: http://scrum.jeffsutherland.com/2010/08/mike-beedle-on-early-history-of-scrum.html

[4] M. Beedle, M. Devos, Y. Sharon, K. Schwaber, and J. Sutherland, "Scrum: A Pattern Language for Hyperproductive Software Development," in *Pattern Languages of Program Design*. vol. 4, N. Harrison, Ed., ed Boston: Addison-Wesley, 1999, pp. 637-651.

[5] J. O. Coplien and N. Harrison, *Organizational patterns of agile software development*. Upper Saddle River, NJ: Pearson Prentice Hall, 2005.

[6] ScrumPloP. (2013). *Scrum Pattern Community*. Available: http://scrumplop.org

[7] K. Schwaber and J. Sutherland, "The Scrum Guide: The Definitive Guide to Scrum, The Rules of the Game," in *Software in 30 Days*, ed: John Wiley & Sons, 2011.

[8] J. Sutherland and K. Schwaber, *The Scrum Papers: Nuts, Bolts, and Origins of an Agile Method*. Boston: Scrum, Inc., 2007.

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

[10] J. Sutherland, A. Viktorov, J. Blount, and N. Puntikov, "Distributed Scrum: Agile Project Management with Outsourced Development Teams," presented at the HICSS'40, Hawaii International Conference on Software Systems, Big Island, Hawaii, 2007.

[11] J. Sutherland, G. Schoonheim, and M. Rijk, "Fully Distributed Scrum: The Secret Sauce for Hyperproductive Offshored Development Teams," in *Agile 2008*, Toronto, 2008.

[12] C. Jones, "Development Practices for Small Software Applications," Software Productivity Research 2007.

[13] C. R. Jakobsen and J. Sutherland, "Scrum and CMMI Going from Good to Great," in *Agile Conference, 2009. AGILE '09.*, 2009, pp. 333-337.

[14] J. Sutherland, S. Downey, and B. Granvik, "Shock Therapy: A Bootstrap for Hyperproductive Scrum," in *Agile Conference, 2009. AGILE '09.*, 2009, pp. 69-73.

[15] J. Sutherland and I. Altman, "Take No Prisoners: How a Venture Capital Group Does Scrum," in *Agile 2009*, Chicago, 2009.

[16] J. Sutherland. (2013). *Teams that Finish Early Accelerate Faster*. Available: https://sites.google.com/a/scrumplop.org/published-patterns/retrospective-pattern-language/teams-that-finish-early-accelerate-faster

[17] P. L. o. G. Process. (2013). *What is a Pattern Language?* Available: http://grouppatternlanguage.org/What_is_a_Pattern_Language

[18] J. Coplien. (1995). *Generative Pattern*. Available: http://c2.com/cgi/wiki?GenerativePattern

[19] J. Sutherland, "Future of Scrum: Parallel Pipelining of Sprints in Complex Projects," presented at the AGILE 2005 Conference, Denver, CO, 2005.