



[Print](#)  
[Close](#)

## Hyper-Productive Agile

Ryan Shriver

April 26, 2010

Today it is generally recognized that agile teams outperform traditional “waterfall” teams with respect to productivity and time to market. What is less well-known is that while some agile teams are 25 to 50 percent more productive than their traditional peers, others are 400 to 800 percent more productive.

So what are these hyper-productive teams doing differently to achieve such extraordinary results? This article introduces you to the key practices used by Scrum teams around the world to achieve hyper-productive results--practices your team can apply right now.

### Productivity and Hyper Productivity

There is no one single, agreed upon definition and measurement of productivity. In this article we use the term to generally mean “[the efficiency of a team at turning inputs into useful outputs](#)”.

With respect to measurement of productivity within software development, there are a few generally accepted ways to measure productivity including [function points](#) and lines of code. Although each of these has their own pros and cons, they do allow some degree of comparison between projects. This is useful for drawing distinctions between those teams that are better than their peers.

Hyper-productivity describes a *state of being* where teams are working at much higher levels of performance such as two, three and four times more productive than their peers. In the past two decades, research has emerged about agile (Scrum most prominently) teams achieving hyper-productive results through the use of agile methods. But these teams are not just doing Scrum-by-the-book, they are interjecting additional practices (primarily engineering in nature) that help them achieve such breakthrough results. But before we talk about hyper-productive teams, let’s see how “normal” agile teams compare to their traditional counterparts.

### Productivity Comparison: Traditional vs. Agile

I think my friend [Dr. David Rico](#) has some of the best research on the [benefits of agile methods](#). Based on his most recent work, the average improvements of agile vs. traditional are:

Metric	Agile Results
Productivity	67% improvement
Quality	65% improvement
Cost	49% reduction

But where David's research spans hundreds if not thousands of projects, [Gabrielle Benefield](#) has published research about [rolling out agile at Yahoo](#) over three years and 150 teams with similar productivity results. Some of the highlights:

<b>Metric</b>	<b>Agile Results</b>
Productivity	<b>34%</b> improvement non-coached teams
Productivity	<b>100% - 200%</b> <a href="#">improvement</a> coached teams
Cost	<b>\$1.5m</b> annual savings per Scrum Coach
Popularity	<b>81%</b> of teams would use Scrum again

Benefield's research, like Rico's, is based upon survey questions about how much *more productive* the teams had become, in the opinion of the respondent. So whereas these results show broad consensus across numerous industries and organizations that agile is much more productive, specific figures vary.

### **Productivity Comparison: Hyper Productive Agile**

But what if we wanted a little more quantified data to support our research? Interestingly enough, it's here that we start to see distinctions between the merely more productive agile teams and those that are hyper productive. [Dr. Jeff Sutherland](#) has likely spent the most time reporting on the subject of hyper-productive agile teams (Scrum specifically) and their results. His findings include:

*A Scrum rollout at [Systematic](#) to reach CMMI-Level 5 resulted in a 100 percent increase in overall productivity in the first six months of implementation. Systematic reported the first doubling of velocity (productivity) resulted from getting software "done" at each Sprint's end (including passing all acceptance tests and resulting in a potentially shippable product). They reported their second doubling of velocity when the Product and Sprint backlogs and user stories were in the "Ready" state at the start of the iteration. This included having the Backlogs prioritized and the acceptance criteria defined and reviewed in advance of the upcoming iteration.*

Two globally distributed teams used these plus engineering practices (sometimes called XP practices) to achieve hyper productive results. The table below shows a comparison of traditional "waterfall" productivity compared to agile using Function Points and Java:

	<b>Waterfall [1]</b>	<b>Waterfall [2]</b>	<b>Scrum [3]</b>	<b>SirsiDynix [4]</b>	<b>Xebia [5]</b>
Person Months	540	2,000	54	827	125
Lines of Java	58,000	500,000	51,000	671,688	100,000
Function Points	900	10,000	959	12,673	1887
Function Points per Developer Month	1.7	5	17.8	15.3	15.1
(Productivity)					

As the highlighted productivity results show, projects of different size and scopes using Scrum and hyper productive practices can achieve relatively similar results three to eight times more productive than their peers--and they are doing this with the same technology as their peers (Java in this case).

In presentations where Jeff Sutherland introduces this information, he reiterates the point that these aren't the run-of-the-mill Scrum teams struggling to get software "Done-Done" each Sprint (a.k.a. iteration). Rather, they've embraced specific practices that are directly leading to their astounding results. So what are these hyper productive practices?

## 12 Practices of Hyper-Productive Teams

The table below contains the practices attributed by Sutherland and others (including my experience in coaching teams) as contributing to hyper productive team performance:

12 Practices of Hyper Productive Teams	
Self Organization	Visible Progress
Engineering Practices	Frequent Communication and Collaboration
Everyone Trained	Pair Immediately
Software is done at Iteration End	Few Disruptions
Backlog is ready at Iteration Start	Everything is Prioritized
Short Iterations (1-2 weeks)	Servant Leadership

While the individual practices are simple, the difference with the hyper-productive teams is their ability to stick to the practices over the project while continually removing impediments limiting performance. This practical use of discipline amongst the team, combined with simple practices, can contribute to break-through performance.

Perhaps one of the most important is self-organization, which Sutherland credits with achieving four to eight times the improvement in productivity. This is the practice where cross-functional teams work together to help each other choose the right work and implement it in as efficient a manner possible. They commit to a Sprint goal together and work aggressively to remove impediments to their progress. The team collectively focuses on the highest priority story for the Sprint and pairs immediately to implement it and test it before moving on to the next story. This is in contrast to agile teams, where individual team members sign up for stories and work primarily by themselves for most of the iteration--only to deluge QA at the end and leave carry-over stories and software not done at Iteration's end.

The use of solid engineering practices is another thing that sets apart hyper-productive teams. The following list is the primary engineering practices the teams adopt:

1. [Continuous Integration](#)– Automatically creating a new build that passes all unit tests after each change to the system
2. [Automated unit testing](#)– Automating developer-written tests that run quickly to ensure changes to the system are completed in safe steps
3. [Automated acceptance testing](#) – Automating QA-written tests that run on-demand and nightly (automated regression) to ensure changes to the system didn't break previous functionality
4. [Push-Button Releases](#) – The practice of being able to run one command and have the latest stable build completely deployed to a defined target environment without human intervention. Having everything automated to push out a new release greatly reduces the overhead and frees up more time

for productive work.

5. [Refactoring](#) – The practice, used in conjunction with unit testing, to make structural changes to the software without changing external features. This is done as part of evolving the architecture as development proceeds each Sprint.

Other practices such as using [test-driven development](#) and [implementing in vertical slices](#) contribute as well, but these are some of the most important ones. Today, tools (predominantly free) exist for implementing these practices in modern technologies such as Java, .NET, Ruby and many others.

The two practices that tend to have the most immediate benefit on getting to hyper productivity are ensuring Software is done at iteration's end followed by ensuring the backlog is ready at iteration's start. The former is often tough for new agile teams, but the key is ensuring that at the end of each iteration you have:

1. Working software that's demonstrated to leaders
2. All unit and acceptance tests are passing
3. Potentially shippable product

This is often referred to as “done-done”. While it takes additional resources to roll this out into production (or ship it if you are a vendor), the software is in a state where it could be done. Hyper-productive teams achieve this at the end of every Sprint and thus always ship on time.

The counterpart to this software is done is to ensure few disruptions. This in turn helps to ensure the backlog is ready at iteration's start. This means each story scheduled for the upcoming iteration is in a ready state:

1. Upcoming stories are identified, prioritized and estimated
2. All acceptance criteria has been defined and reviewed with customer, developers and testers (oftentimes just the lead--not the entire team)
3. Developers, analyst and testers have conversation about the story before development begins to flush out any details and ensure consistent understanding

When practicing short iterations (1-2 weeks), this practice becomes a necessity because any delay--even a few hours--can impact the ability to complete a story within a Sprint. While minor details can be worked out within the Sprint, ensuring all the scenarios (positive and negative) have been thought through in advance and other major details must be worked out in advance.

For stories that don't pass the ready-state checklist by iteration start, they simply aren't scheduled. Doing so ensures the team remains fully productive during the Iteration and doesn't squander time waiting on decisions that could have been made in advance. But the advantage with Sshort iterations is the rapid feedback from the customer validating the team is moving in the right direction. Coupled with the team cadence achieved in this state, working in short, defined steps helps teams develop a rhythm for their development.

Visible progress is where the team leverages [story boards](#) to show the stories (and tasks) currently under development and their related state. [Burndown charts](#) are used to show the progress a team is making to getting stories complete over the course of a release and/or Sprint. These [information radiators](#) help facilitate frequent communication and collaboration on getting stories done with simple means of communicating key information to all team members. But above the tools, the primary purpose of frequent communication and collaboration is to ensure the team works through problems together and uncovers obstacles in advance to

pave the way for quick implementation of stories. The [daily stand-up meeting](#), [pair programming](#) and tools such as Wikis, instant messenger and continuous integration all contribute to ensuring teams can work rapidly with few disruptions and wide-open channels of communication.

To ensure the quickest implementation of each new story, hyper-productive teams practice pair immediately when starting any new story. That is, two developers work together (side-by-side) on one computer to implement the story. Just like an airliner has a pilot and co-pilot/navigator, two developers can work much quicker as a unit than they can independently. Once teams get into the rhythm of developing in pairs, not only does it take less time to develop new stories, the stories also get done faster thus enabling short iterations (1-2 weeks). Pair immediately has some advantages for risk mitigation including ensuring knowledge is shared amongst the team and conventions are consistently implemented. In addition, pairing helps mitigate overdependence on key resources that can have a negative impact on productivity.

Within each iteration the team needs to remain focused on the stories at hand and therefore needs to have few disruptions. This includes changing of priorities within the iteration or receiving multiple conflicting priorities from stakeholders. Therefore, hyper-productive teams ensure all changes flow through the [product owner](#) at the right time (iteration start). In addition, the team works to ensure non-planned events such as production outages or issues are kept to a minimum--or at least handled in a way that minimizes the disruption to the entire team. One common example is one analyst (and/or developer) doing triage on all incoming issues and working with the product owner to establish priorities of these with the other stories scheduled.

One of the responsibilities of the product owner is to ensure everything is prioritized; this includes the stories within the [product backlog](#) and [Sprint backlog](#). In doing so, the team always knows the most important thing to work on next ensuring the team can remain focused on what's most important.

A final key practice for hyper-productive teams is to have servant leaders such as the agile coach and product owner--who both provide team leadership but also work for the team to actively remove impediments for their progress. This has the effect of not only improvement productivity for the team but also helping ensure morale stays higher because the team can focus on delivering stories.

## Summary

Together, these practices--including insuring everyone is trained in agile--can have a tremendous benefit to not only productivity but also the enjoyment from working in a team that's jelled. While the practices are simple in concept, those teams that stick with them have historically performed at levels that are hyper productive and far exceed their peers.

[Ryan Shriver](#) is a Managing Consultant with [Dominion Digital](#), a Virginia-based process and technology-consulting firm. Based in Richmond, he leads the [IT Performance Improvement Solution](#), which includes [Agile Adoption](#), [Agile Engineering](#), [IT Process Improvement](#) and [IT Services Management](#). With a background in systems architecture and large-scale agile development, Ryan currently focuses on measurable business value and systems engineering. He writes and speaks on these topics in the US and Europe, posting his current thoughts at [theagileengineer.com](#). Ryan can be reached at [rshriver@dominiondigital.com](mailto:rshriver@dominiondigital.com).

---

[1] From [User Stories Applied for Agile Development](#) by Mike Cohn (2004)

[2] From [Software Assessments, Benchmarks and Best Practices](#) by Capers Jones (2000)

[3] From [User Stories Applied for Agile Development](#) by Mike Cohn (2004)

[4] Distributed Scrum: Agile Project Management by Outsourced Teams by J. Sutherland, A. Viktorov, J. Blount, N. Puntikov (2007)

[\[5\]](#) Fully Distributed Scrum: The Secret Sauce for Hyperproductive Outsourced Development Teams by J. Sutherland, G. Schoonheim, E. Rustenburg, M. Rijk (2008)

Copyright © 2010 gantthead.com All rights reserved.

The URL for this article is:  
<http://www.gantthead.com/article.cfm?ID=255949>