

Extracted from:

# The Agile Samurai

---

## How Agile Masters Deliver Great Software

This PDF file contains pages extracted from The Agile Samurai, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

**Note:** This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2010 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The  
Pragmatic  
Programmers

# The Agile Samurai

How Agile Masters  
Deliver  
Great Software



**Jonathan Rasmusson**

*Edited by Susannah Davidson Pfalzer*

# Setting Up a Visual Workspace

---



Flight status boards are great. In one quick glance you can see what's coming, what's going, and what's been canceled altogether.

Why not do the same for your project?

By learning how to create a visual workspace, you and the team will never be at a loss for what to do next or where you can add the greatest value. Not only will this enable you to work with greater clarity and focus, the increased transparency will also help you set expectations with the powers that be.

Speaking of which, here they come now.

## 11.1 Uh-oh...Here Come the Heavies!

There's been a big shake-up at corporate. Budgets have been cut. Timelines have been slashed. And everything needs to be done better, faster, and cheaper.

As a result, you've been asked to do more with less. Management would like you to deliver the same amount of functionality, with half the team, one month ahead of schedule. Or else.

It's all coming down hard and fast, and tomorrow they want to set up a meeting with you to confirm you are on board with the new plan.

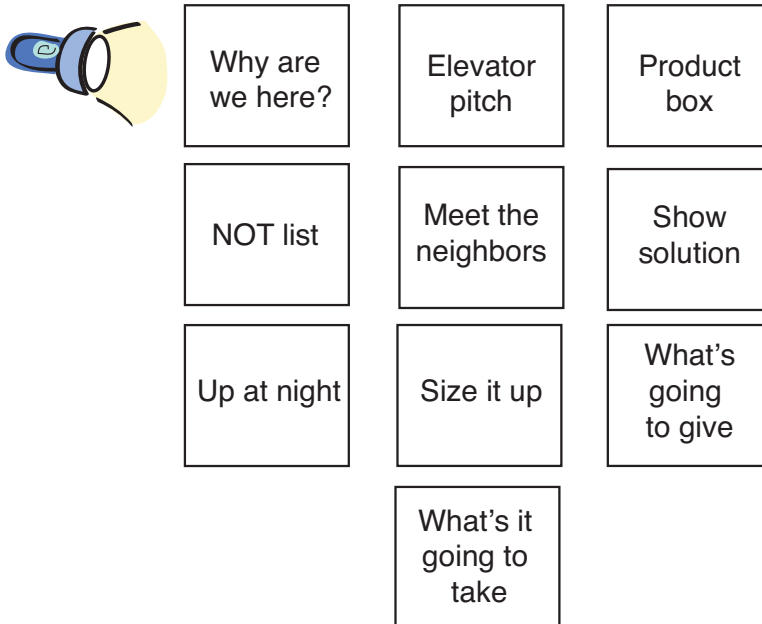
Gulp! What do you do? What they are asking for is completely unreasonable. You know it. The team knows. It seems they are the only ones who don't.

What could you do to show that while you would love nothing more than to be able to deliver the same amount of functionality with half the resources, it ain't gonna happen.

### Bringing the Executives Up to Speed

Instead of setting up a formal meeting and pleading your case in PowerPoint, you invite the executives down to your work area to see firsthand the state of the project.

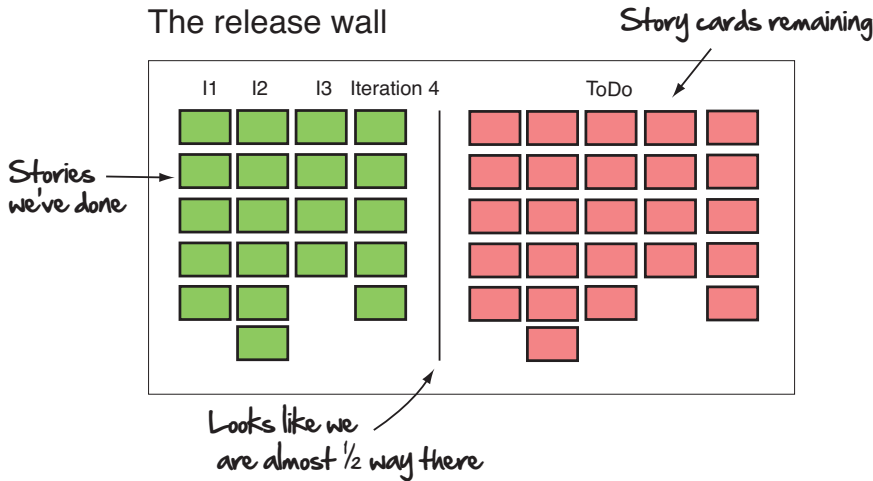
You begin by taking them through the inception deck for your project, which you conveniently have posted on the wall.



The inception deck, you explain, is a tool you and the team use to make sure you never lose sight of the goal of the project. By making it visible,

you always know who the customer is, what they're after, and, most important, why we decided to spend money on this project in the first place.

Impressed, the executives lean closer and ask you where you are in the project. To answer that, you then direct their attention to your *release wall*.



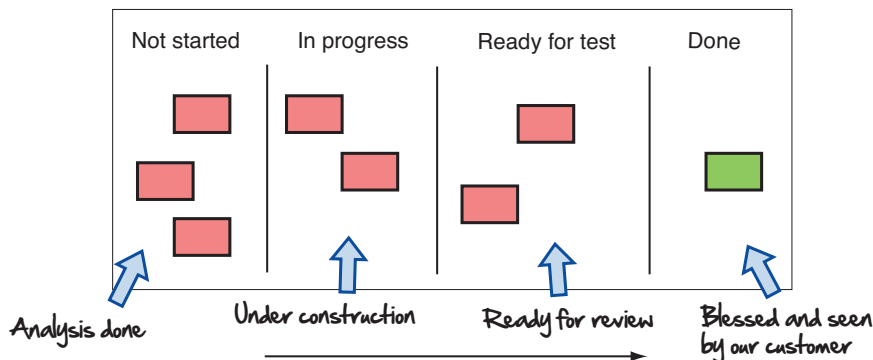
The release wall is where you and the team track keep track of what's been done and what's remaining. The left side of the wall shows those features that have been fully analyzed, developed, tested, and vetted by the customer (they are ready to be shipped). And the right side shows those stories still needing to be developed.

As far as what the team is working on this iteration, you draw management's attention over to this iteration's storyboard.

## The storyboard

Current iteration

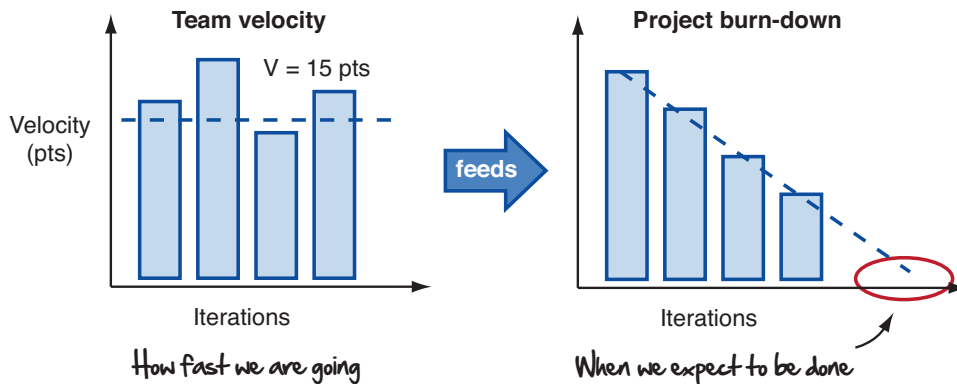
State of this iteration's user stories



The storyboard tracks the state of this iteration's features (or what we call *user stories*). Features yet to be developed live on the left, while those that have been built and blessed by the customer live on the right. As a story gets more developed, it moves across the board from left to right. Only when it is fully developed, tested, and vetted by the customer does it get moved into the Done column.

Looking at their watches, they then cut to the chase and ask when *you* expect to be done.

To answer that, you bring them over to the only two charts on your wall you haven't shown them yet—your team velocity and the project burn-down chart.



You explain that the team velocity is the closest thing you and the team have for measuring the team's level of productivity. By measuring how much the team gets done each week and using that as the basis of planning going forward, the team can accurately predict when they expect to be done. This is shown on the project burn-down chart.

The project burn-down (Section 8.5, *The Burn-Down Chart*, on page 150) takes the team velocity and extrapolates the speed at which the team is "burning" through the customer's wish list. The project is done when the team delivers everything on the list or the project runs out of money (whichever comes first).

With the stage set, you now calmly point out what should already be obvious to everyone in the room. Halving the development team would effectively cut the team's productivity in half.

Impressed with your command of the situation, the executives thank you for your time and move onto their next project meeting.

A few weeks later you get an email explaining that because of the company's heading in a new strategic direction, your project is going to be canceled (life is like that sometimes).

The good news, however, is that they were so impressed with how you managed your project, they want you to play a lead role in the new initiative!

This is just one contrived example of how a visual workspace can help you set expectations with stakeholders and make the reality of a situation self-evident. But where it really shines is in helping you and your team execute and focus.

Let's now go over some ideas for creating your own visual workspace.

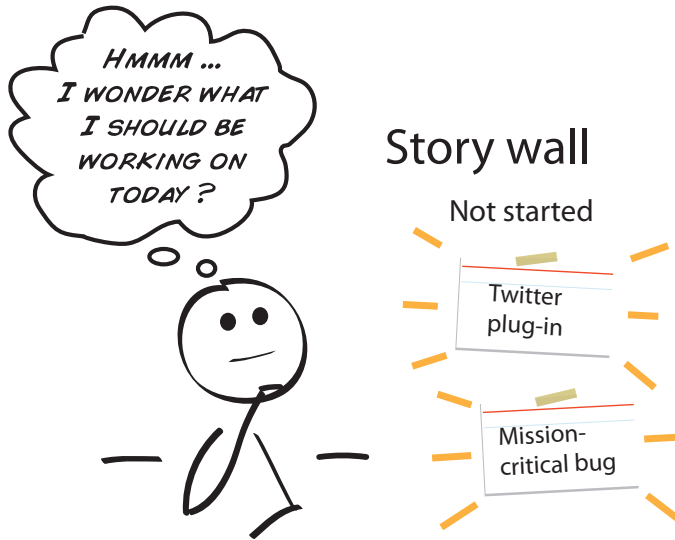
## 11.2 How to Create a Visual Workspace

Creating a good visual workspace is pretty straightforward. For teams new to agile, I usually recommend starting with the following:

- A story wall
- A release wall
- A velocity and burn-down graph
- An inception deck, if they have the room

The inception deck is good because it reminds the team why they are there and what it's really all about (which can be easy to lose sight of when your head is buried in your project).

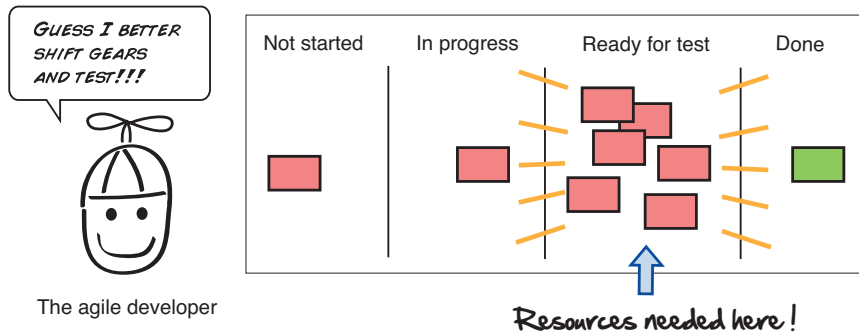
The story wall is great because any morning anyone can walk in and know exactly what needs to be done next.



The story wall will also show you any bottlenecks you have in the system and where you'll want to direct resources.

### The storyboard

Current iteration



The release wall is a thing of beauty, because anyone can walk into your room and see the state of your project at a glance. This is what's done. This is what's remaining. No fancy math or Excel spreadsheets required.

And as we talked about extensively in agile planning, nothing sets expectations better than a good burn-down chart. Keep one of these babies on your wall, and you'll always know how realistic your dates are looking and how you are trending.

And of course this is just the beginning. If you have other pictures, mock-ups, or diagrams that help you and your team execute, stick 'em up there and make it visible for all to see.

Here are some other ideas for creating your visual workspace.

### 11.3 Show Your Intent

Working agreements are about putting a stake in the ground as a team and saying, “This is how we as a team like to work.” It’s a way of setting expectations with everyone on the team about how your team is going to work and what’s going to be expected of people if they join you on this ride.

Shared values are the same, only more touchy-feely. If the team has been burned in the past because they were forced to compromise on quality and no longer want to be known as that team that cuts corners and writes crappy software, they can post their shared values and make that known.

Working agreements	Shared values
* Core hours 9 a.m.-4 p.m.	* We don't cut corners
* Daily stand-ups 10 a.m. sharp	* No broken windows
* Done includes testing	* It's OK to disagree
* Respect the build	* We can handle the truth
* When someone asks you for help say “yes”	* Don't assume—ask
* Weekly demo Tues 11am	* When in doubt—write a test
* Customer available 1-3 p.m.	* Crave feedback
	* Check your ego at the door

The other thing you want to be sure you share on your project is language.

## 11.4 Create and Share a Common Domain Language



When the words used in your software don't match those used by business, you can get into all sorts of trouble.

- The wrong abstractions get built into the software (business will think *location* means one thing, while developers will interpret it to mean something else).
- The software becomes harder to change (because the words that appear on the screen don't match those used to store it in the database).
- You end up with more bugs and higher maintenance costs (because the team has to work extra hard when making changes to the software).

To avoid this dysfunction, create a common language that you and the business share and use it relentlessly in your user stories, models, pictures, and code.

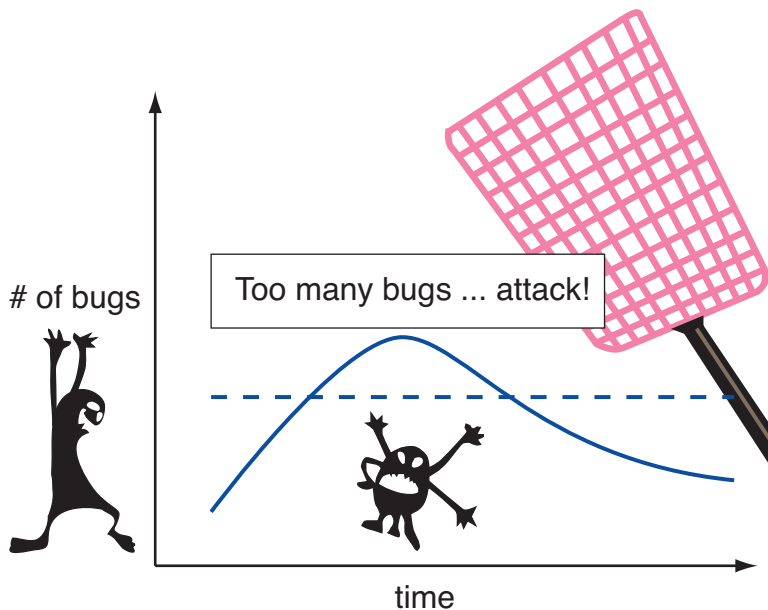
For example, if there are some key words that you and your customer use when you talk about the system, write them down, come up with clear definitions about what these words mean, and then make sure you match those definitions on the software (that is, screens, code, and database columns).

Doing this will not only minimize the bugs and rework but also make it way easier to talk to your customer because your code will always be in lockstep with how they talk about their business.

We don't have the time or space to do this topic justice. But there is an excellent book on the subject by Eric Evans: *Domain-Driven Design: Tackling Complexity in the Heart of Software*. [Eva03]. It's well worth the read.

Finally, watch your bugs.

## 11.5 Watch Those Bugs



To make sure you and your team aren't overwhelmed by a surprise bug attack just before you roll into production, track and keep your bug count down from day one of your project.

If it helps, dedicate 10 percent of every iteration to bug squashing and paying down technical debt. Just squash those buggers on the spot, and don't let that bug count get away from you.



## Master Sensei and the aspiring warrior

**STUDENT:** *Master, what if my workplace does not allow me to create a visual workspace? What should I do?*

**MASTER:** *It is true that some office work environments resist project teams putting their work artifacts up on the wall. When faced with this resistance, accept that it is there and decide how to proceed.*

**STUDENT:** *Yes, Master. But should I fight for the visual workspace? Or just accept that I can't have one?*

**MASTER:** *That is up to you. You can compromise. You can acquiesce. Or you can confront. There is a time and place for each. Search your heart, seek allies, and decide whether this battle is worth the effort.*

**STUDENT:** *If this is truly an important practice, what can one do to compromise?*

**MASTER:** *When faced with situations such as these, some warriors have found creating fold-away storyboards useful for keeping the workplace clean, while enabling the team to communicate openly during the day. Others have used online tools and virtual storyboards for sharing important information, as well as keeping the team in sync.*

**STUDENT:** *So, my visual workspace doesn't always have to be physical?*

**MASTER:** *No. Physical is best but sometimes not always possible.*

**STUDENT:** *What if I choose to confront? What should I do then?*

**MASTER:** *You can start by simply creating a visual workspace, use it daily for your project, and hope that, through dialogue and education, the benefits become self-evident.*

**STUDENT:** *And if they do not?*

**MASTER:** *Then the root cause is usually one based on emotion. There may be forces diametrically opposed to what you are trying to achieve. Try empathizing and understanding the spirit behind those forces arrayed against you. Perhaps through dialogue, you will be able to find a solution that works for both parties. Time and patience may be your best allies here.*

### **What's Next?**

Your journey is almost complete. You've got everyone on the bus (Chapter 3, *How to Get Everyone on the Bus*, on page 50), you've got the plan (Chapter 8, *Agile Planning: Dealing with Reality*, on page 132), and you know what it takes to execute.

The next part of the book, "Agile Software Engineering," focuses on the core agile software engineering practices you and your team are going to need to make all this agile stuff happen.

It's a must-read if you cut code, but it is also recommended if you ever plan on leading an agile project. None of this agile stuff works unless it's backed by some really solid technical practices, and although each of the next four chapters could be a book in themselves, the chapters here will give you enough of a taste to understand how the practices work and why they are important for your team's overall agility.

We'll start by taking a look at one of the greatest time-savers of any software project—automated unit testing.

# The Pragmatic Bookshelf

---

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

## Visit Us Online

---

### **The Agile Samurai**

<http://pragprog.com/titles/jtrap>

Source code from this book, errata, and other resources. Come give us feedback, too!

### **Register for Updates**

<http://pragprog.com/updates>

Be notified when updates and new books become available.

### **Join the Community**

<http://pragprog.com/community>

Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

### **New and Noteworthy**

<http://pragprog.com/news>

Check out the latest pragmatic developments, new titles and other offerings.

## Buy the Book

---

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: [pragprog.com/titles/jtrap](http://pragprog.com/titles/jtrap).

## Contact Us

---

Online Orders:	<a href="http://www.pragprog.com/catalog">www.pragprog.com/catalog</a>
Customer Service:	<a href="mailto:support@pragprog.com">support@pragprog.com</a>
Non-English Versions:	<a href="mailto:translations@pragprog.com">translations@pragprog.com</a>
Pragmatic Teaching:	<a href="mailto:academic@pragprog.com">academic@pragprog.com</a>
Author Proposals:	<a href="mailto:proposals@pragprog.com">proposals@pragprog.com</a>
Contact us:	1-800-699-PROG (+1 919 847 3884)