

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

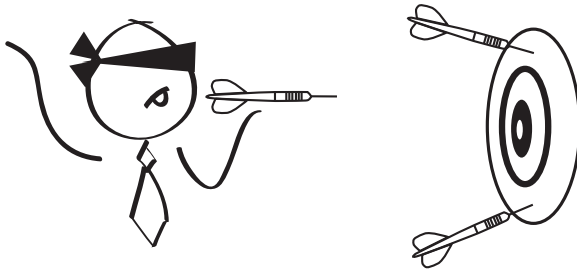
How Agile Masters
Deliver
Great Software



Jonathan Rasmusson

Edited by Susannah Davidson Pfalzer

Estimation: The Fine Art of Guessing



Get ready to bring some reality back to the estimation process. Agile dispenses with the pleasantries and reminds us what our initial high-level estimates really are—really, they're bad guesses.

But by learning how to estimate the agile way, you'll stop trying to get something your up-front estimates can't give (precision and accuracy) and instead focus on what really matters—building a plan you and your customer can work with and believe in.

In this chapter, you'll learn how to estimate your user stories the agile way, as well as some powerful group estimation techniques for sizing up things.

7.1 The Problem with High-Level Estimates

Let's face it. Our industry has had some challenges when it comes to setting expectations around estimates on software projects.

The Point of Estimation

"The primary purpose of software estimation is not to predict a project's outcome; it is to determine whether a project's targets are realistic enough to allow the project to be controlled to meet them."

—Steve McConnell, *Software Estimation: Demystifying the Black Art* (McC06)

It's not that our estimates are necessarily wrong (though they almost always are). It's more that people have looked to estimates for something they can never give—an accurate prediction of the future.

*JOHNSON ! GET ME A
DETAILED ESTIMATE
FOR OUR ...*



*YET TO BE SPEC'D SYSTEM, USING OUR
YET TO BE DETERMINED TECHNOLOGY, WITH OUR
YET TO BE DETERMINED TEAM, IN OUR
YET TO BE DETERMINED BUSINESS ENVIRONMENT
TO BE BUILT NEXT YEAR.*

It's like somewhere along the way, people lost sight of the fact that

HIGH-LEVEL ESTIMATES ARE GUESSES
(AND USUALLY REALLY BAD, OVERLY OPTIMISTIC ONES AT THAT)

And it is when these up-front, inaccurate, high-level estimates get turned prematurely into hard commitments that most projects get into trouble.

Steve McConnell refers to this dysfunctional behavior as the cone of uncertainty (Figure 7.1, on the following page), which reminds us that initial estimates can vary by as much as 400 percent at the inception phase of our project.

The simple fact is that *accurate up-front estimates aren't possible*, and we need to stop pretending they are.

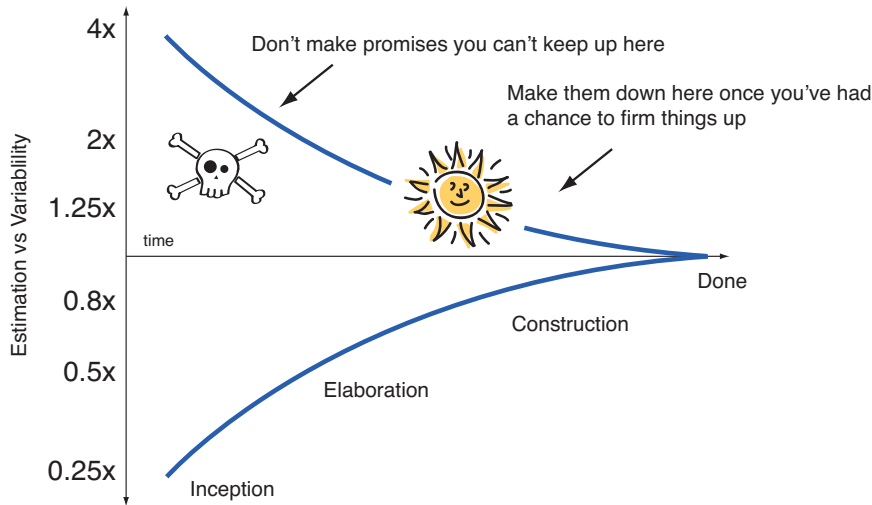


Figure 7.1: The cone of uncertainty reminds us of how greatly our estimates can vary at different stages throughout the project.

The only question our up-front estimates can attempt to answer is this:

IS THIS PROJECT EVEN POSSIBLE !?

(GIVEN THE TIME AND RESOURCES WE'VE GOT)

What we need is a way of estimating that does the following:

- Allows us to plan for the future
- Reminds us that our estimates are guesses
- Acknowledges the inherent complexities in creating software

7.2 Turning Lemons into Lemonade

In agile, we accept that our initial, high-level estimates aren't to be trusted. However, we also understand that budgets need to be created and expectations need to be set.

To make that happen, the warrior does what anyone would do who is looking to firm up any estimate. They build something, measure how long that takes, and use that for planning going forward.

For that to work, we need two things:

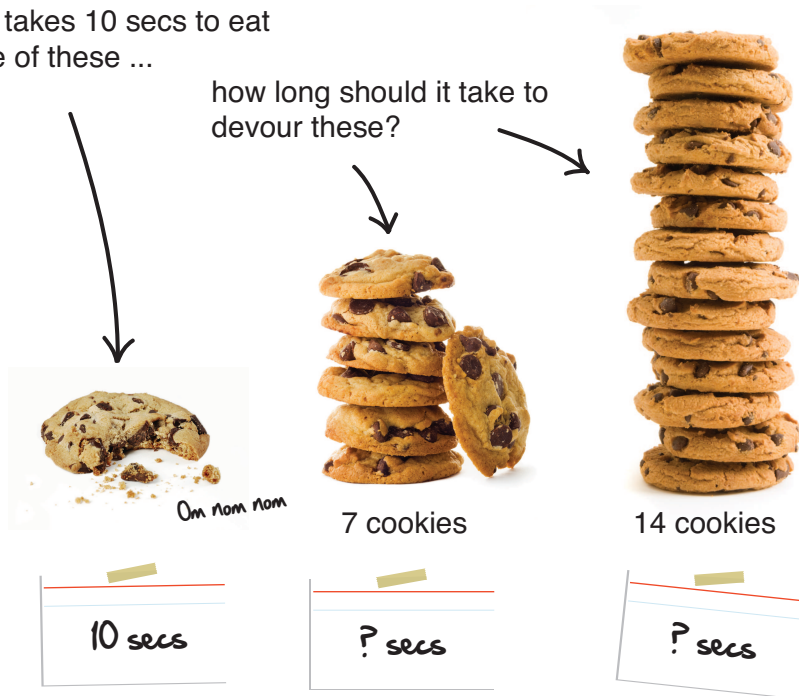
- Stories that are sized *relatively* to each other
- A *point-based* system to track progress

Let's look at each of these in more detail and see how they help us plan.

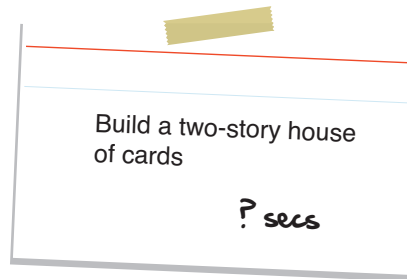
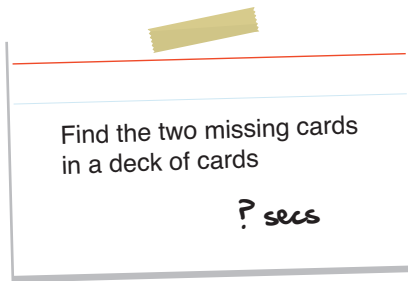
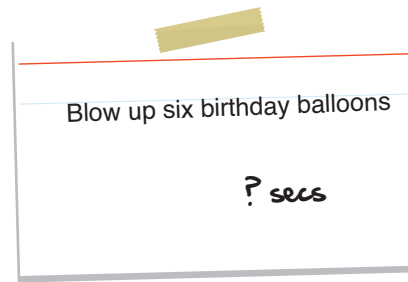
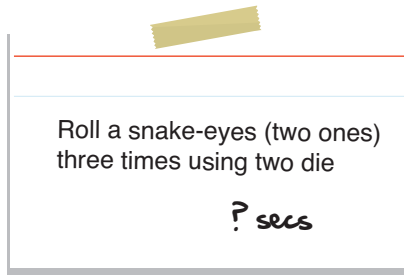
Relative Sizing

Imagine you knew it took ten seconds to eat one chocolate chip cookie, and you were asked to estimate how long it would take you to devour a pile of seven and fourteen cookies (glass of milk included). What would your guess be?

If it takes 10 secs to eat one of these ...



Now imagine you were asked to estimate something else—something simple but maybe something you haven't done many times before. How long do you think it would take you to do these four simple tasks?



If you are like most people, you probably found estimating the cookies relatively easy (pun intended) and the other tasks absolutely harder.

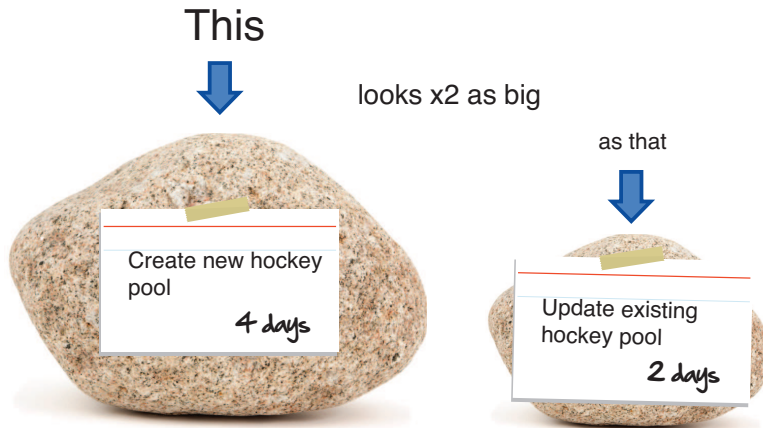
If one cookie = 10 sec

then seven cookies = $10 \text{ sec} \times 7 = 70 \text{ sec}$

and fourteen cookies = $10 \text{ sec} \times 14 = 140 \text{ secs}$ > *x2 as big*

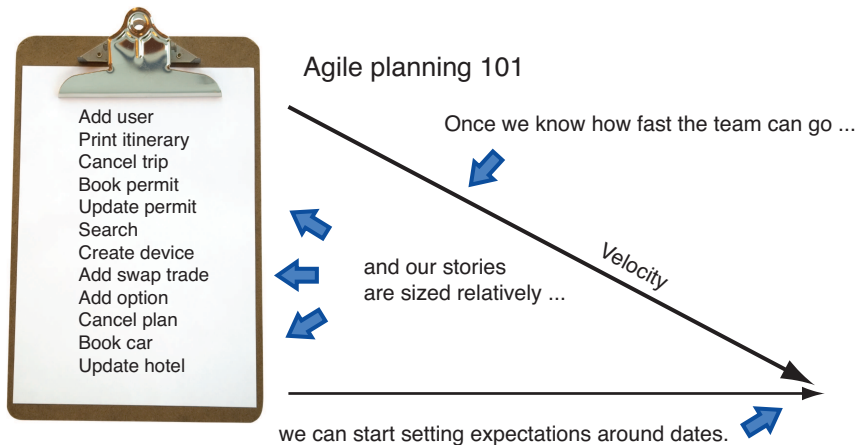
The difference between the two exercises was that with the cookies we estimated relatively while the card counting we estimated absolutely.

Science has shown that estimating relatively is something we humans are actually pretty good at. When you put two rocks in front of us, we can tell quite accurately how much bigger one rock is than the other.



Where we struggle is with telling you precisely how much bigger it is (estimating absolutely).

This simple principle forms the cornerstone of agile estimation and planning. By sizing our stories relatively to each other and measuring how fast we can go, we have all the ingredients we need to begin forming our agile plan.



Now, one challenge with estimating relatively is that a single day in our estimates won't always equal one day in our plans. The team will work either slower or faster than we originally estimated.

1 RELATIVE DAY ≠ 1 calendar day

To account for this discrepancy and avoid continuously having to reestimate all our stories, agile does estimation using a point-based system.

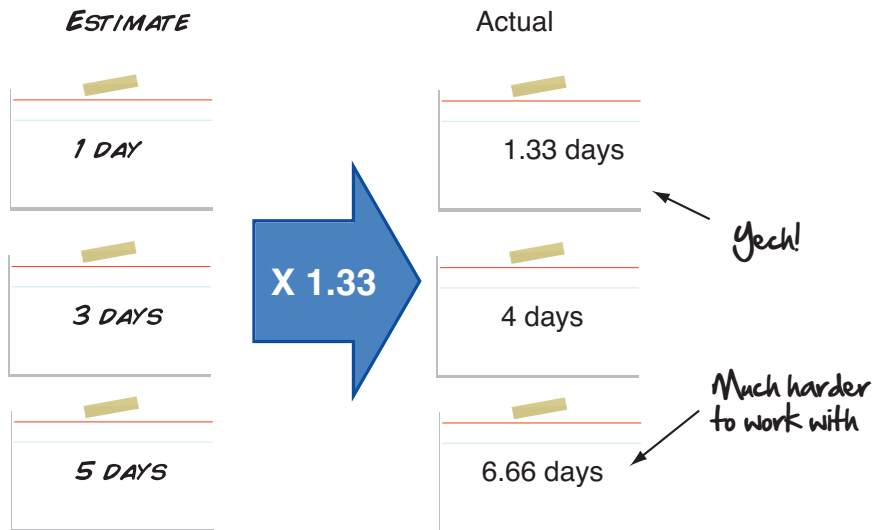
Point-Based Systems

Point-based systems enable us to track progress and estimate relatively without having to worry about how our actuals compared with our estimates.

Say, for example, we originally estimated a story to take three days when in reality it ends up taking closer to four.



We could try to adjust all our estimates by 33 percent.



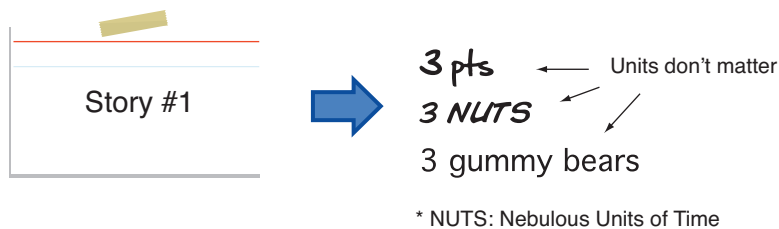
But who wants to work with numbers like 1.33 and 6.66 days? Not only is there a false sense of precision, but what do we do when after delivering a few more stories we find our 1.33 day estimates are closer to 1.66? Readjust again?

To get away from this constant, never-ending rejigging of the numbers, agile recommends freezing your estimates on a simple, easy-to-

use point-based system and not tying them to elapsed time on the calendar.



With a point-based system, our units of measure don't matter. The measure is one of relativity—not absoluteness.

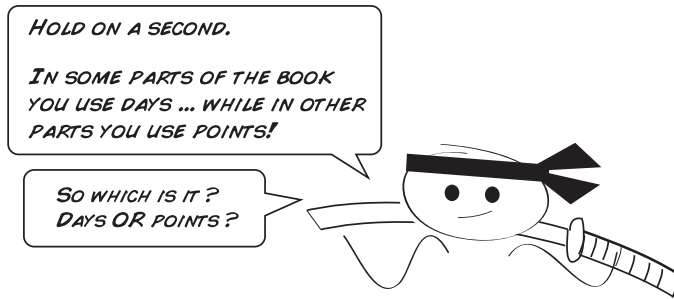


All we are trying to do is capture the *bigness* of a task with a number and size it relatively to all the others. If it helps, you can think of agile estimation as trying to sort your stories according to T-shirt size: small, medium, or large.

Also, as caught up as we tend to get with our estimates, at the end of the day, it doesn't really matter. So long as we size our stories similarly to each other, for every story we over-estimate, there's usually another we under-estimate. So, it all comes out even in the end.

Using a point-based system does the following for us, and studies show we are actually pretty good at it:

- It reminds us that our estimates are guesses.
- It is a measure of pure size (they don't decay over time).
- It's fast, easy, and simple.



You got me. Before we got to agile estimation, whenever the topic of estimation came up, I used the term *days* when really I should have been using *points*. I did this for two reasons. First, we hadn't had a chance to talk about the concept of estimation using *points*. Second, because some agile teams do estimate in days, they just call them something else—ideal days.

Ideal days are just another form of story point. An ideal day is the perfect day where you have no interruptions and are able to work for eight hours straight of uninterrupted bliss.

Of course, we never get ideal days at work, but some teams find the concept useful.

Ideal days can work, but I generally prefer sticking to points. Mostly it's because it makes the fact we are estimating in points explicit, but also because with points I don't have to worry about my ideal day not equaling yours.

For the rest of the book, don't panic if you see points instead of days. I'll stick with points for the remainder of the book, but if you see days, know they are the same thing.

7.3 How Does It Work?

That's enough talk. It's time to get real. Here are two simple estimation techniques you and your team can use to size your stories appropriately for agile planning.

Triangulation

Triangulation is about taking a few sample reference stories and sizing our other stories relatively to these.

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.

Contact Us

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