Reference Class Forecasting

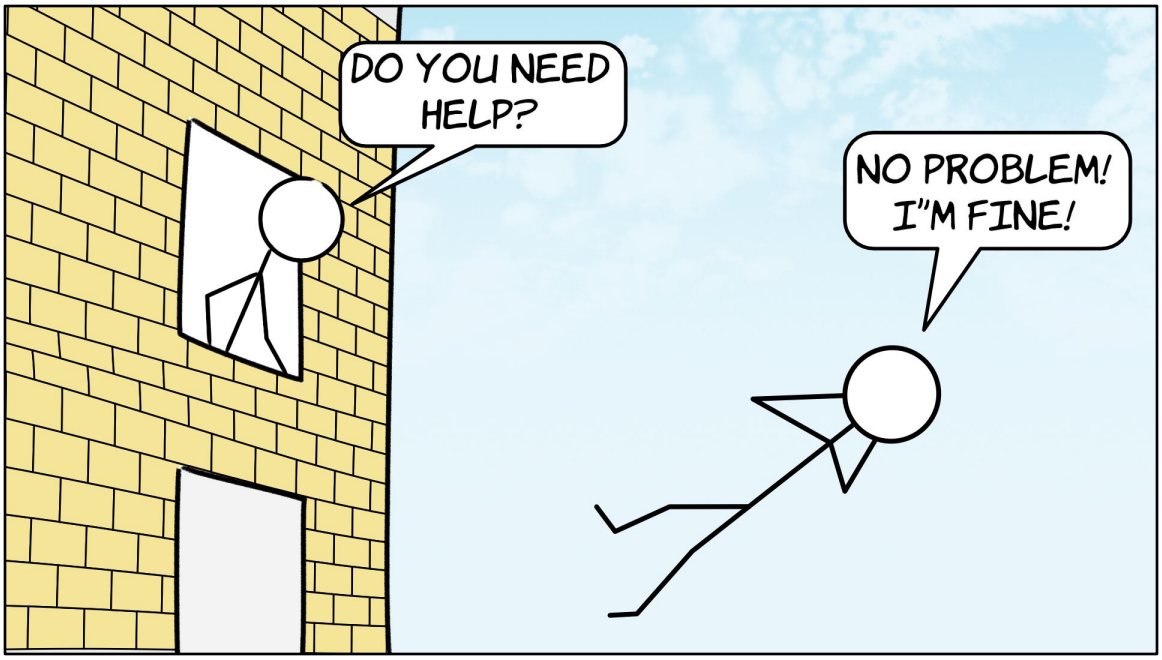
When will it be done?

What will it cost?

By Henrik Mårtensson
self@henrikmartensson.org
+46 76 312 70 68

We are going to begin this talk by going straight for the jugular. We'll talk about time estimates!

[**CLICK!**] Don't worry about the slide! It's just strawberry jam.

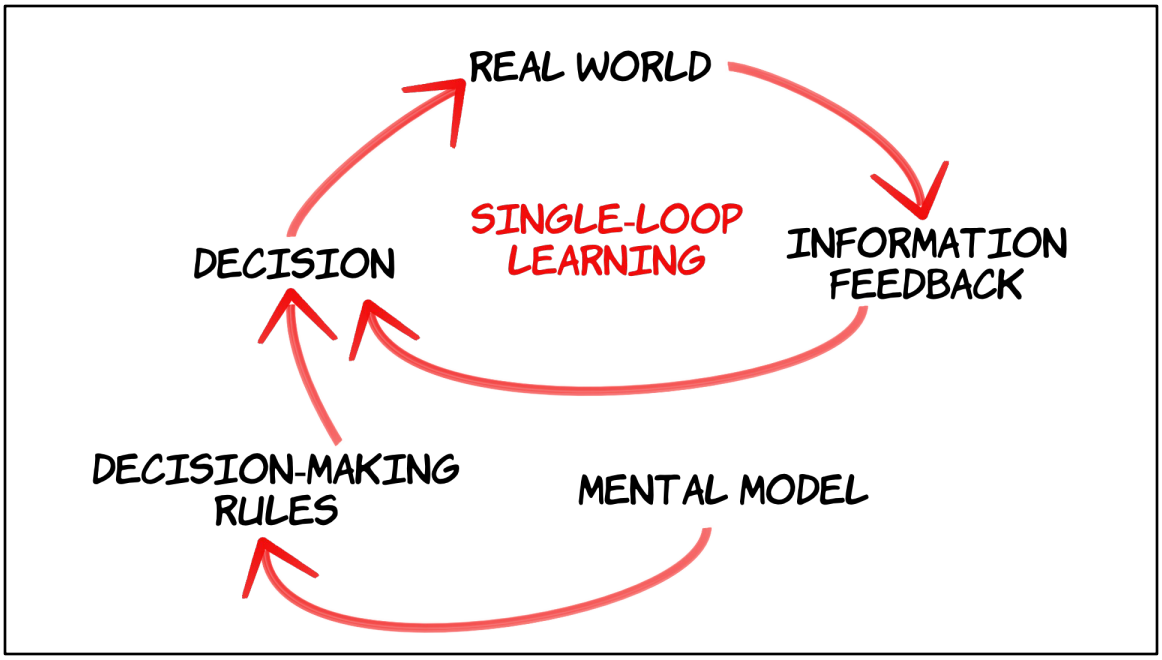


Every software development organization I have worked with the past forty years has had the same problem with time estimates: They are almost always wrong!

It does not matter which method you use to make the estimates: Poker planning, T-Shirt estimates, Critical Path, Critical Chain...they rarely work.

And yet, very few organizations take a serious look at the problem. Instead, they just live with the problem, or they keep switching between different estimation methods, without ever getting reliable results.

In particular, they do not consider whether estimation is the right thing to do, and they do not consider whether there are alternatives.



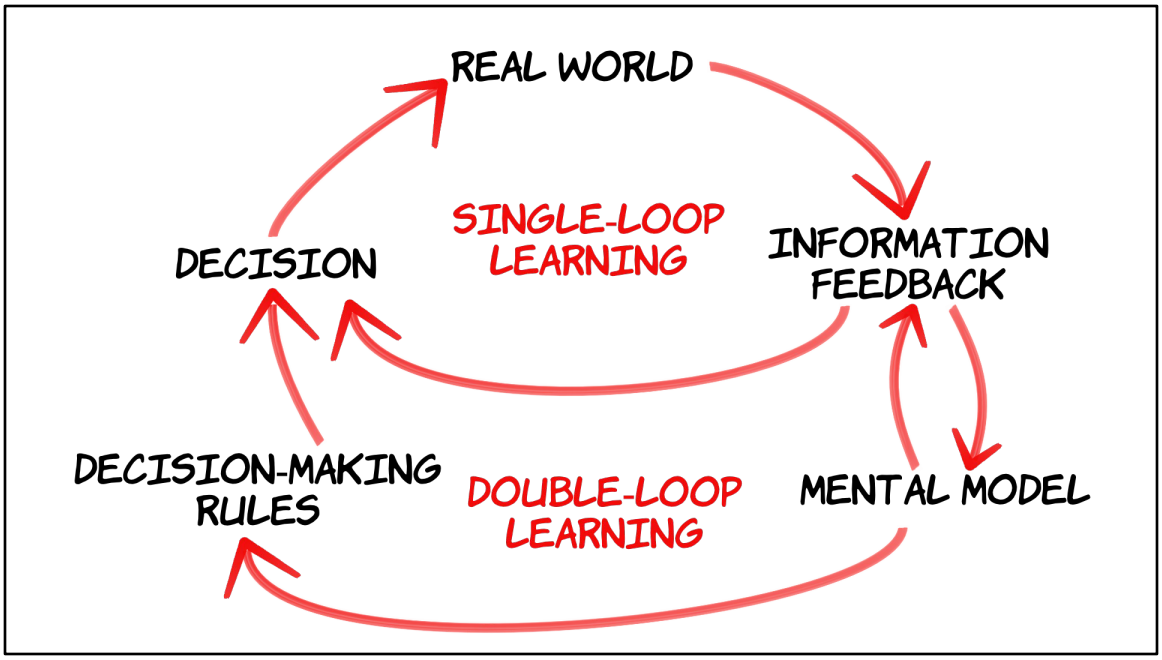
There is actually a name for this, it is called single-loop learning. This model was created by the organizational behavior expert Chris Argyris.

In single-loop learning, we have a mental model of how something works. Based on that model, we have a set of decision making rules.

In this case, we have a mental model that says project duration, and cost, can be predicted by decomposing the project into parts, estimating the parts, and adding the estimates together.

Thus, what we need to do, is to select the best way of making the estimates.

In many cases, single-loop learning works well enough, but if there is something wrong with your mental model, you will never find a good solution.



Chris Argyris came up with another learning model, double-loop learning. In the double-loop model, the feedback we get changes our mental model.

When our mental model changes, our decision-making rules change, and we are suddenly able to look for new solutions to our problem.

Today, we will go from single-loop learning, to double-loop learning, to solve the problem of bad estimates.

Why Do We Need Time Estimates?

- When Will it be Done?
- What Will it Cost?
- Go/No Go Decisions
- People and Resource Allocation
- Synchronizing Activities
- ~~Timebox Activities~~

First, we will ask ourselves why we need estimates. This may seem like a dumb question. It is pretty obvious we need them to predict when something will be done, and what it will cost to get it done.

We will dig a little bit deeper though.

[**CLICK!**] We use estimates to decide if something is worth doing at all. That is, we make Go/No Go decisions.

[**CLICK!**] We use estimates to decide how and when to allocate people and resources.

[**CLICK!**] We use estimates to synchronize activities.

[**CLICK!**] We also use estimates to timebox activities, for example during Sprint Planning in Scrum.

[**CLICK!**] As it turns out, that last one is not really necessary. Projects work better without timeboxed sprints. We will have a look at why in this presentation.

Part 2: Estimates...the Wrong solution?

FOR EVERY COMPLEX PROBLEM, THERE IS A SOLUTION THAT IS CLEAR, SIMPLE, AND WRONG!

-H.L. MENCKEN

The American journalist H.L. Mencken once wrote that “for every complex problem, there is a solution that is clear, simple, and wrong.”

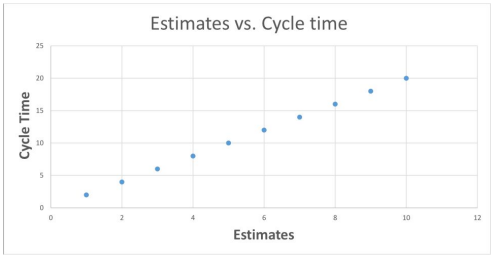
Mencken himself had some questionable views, so he was in a good position to know.

He had a good point though. Sometimes, the clear, simple solution is the wrong solution.

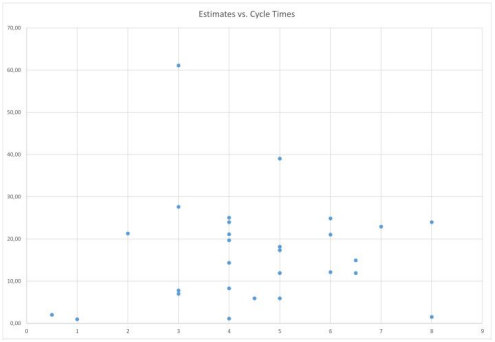
It would be nice if we could find some way to test if estimates are the right solution, or if estimates are one of those wrong solutions.

Part 2: Estimates...the Wrong solution?

What we expect



What we get



All estimation methods, from the simplest, to the most complicated, are based on the idea that we can make a guess about how long a unit of work will take, or what it will cost, and that the guess will be so close to the actual result that it is useful for making decisions.

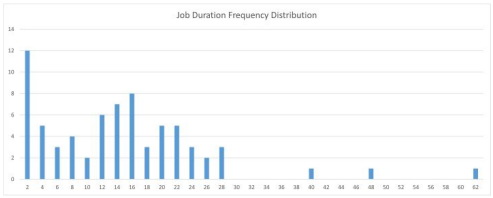
Ideally, if we plot estimates against actual outcomes, in this case cycle time, in a scatter plot, we will get a straight line.

[**CLICK!**] If we look at real data, from a project team, what we will see is almost always very different from what we expect.

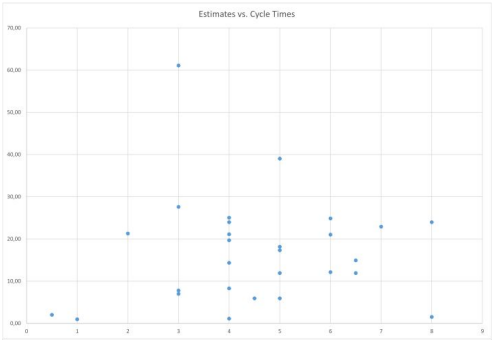
We don't get a line. Instead, we get a cloud of random points.

Part 2: Estimates...the Wrong solution?

Frequency distribution of job durations



Estimates vs. Cycle Times

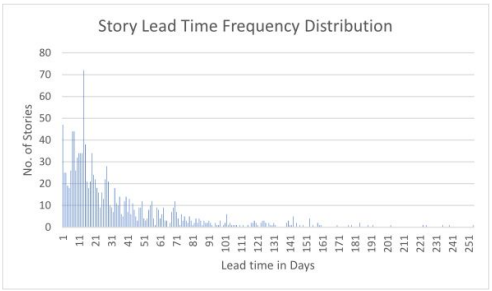


If we build a frequency distribution table of actual job durations, we will get something like this:

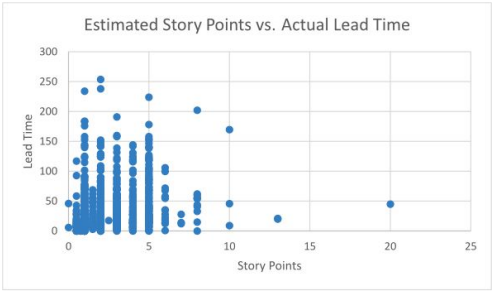
Quite a few jobs are done very quickly, but some of them take a lot longer, and there is a long, drawn out tail that can stretch out a long, long way.

Part 2: Estimates...the Wrong solution?

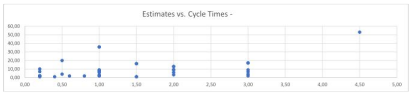
Frequency distribution of job durations

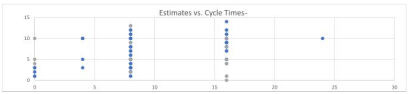


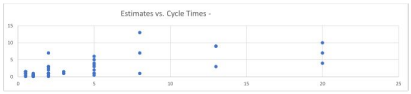
Estimates vs. Cycle Times

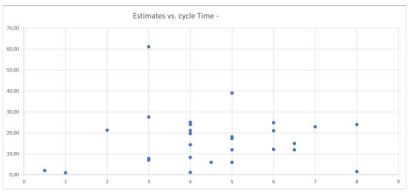


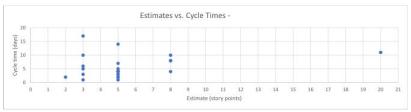
It does not really matter if we look at cycle time or lead time, or which team we look at. Almost all teams have this kind of long tail distribution of job durations, and randomness when you plot estimates against actual outcomes.



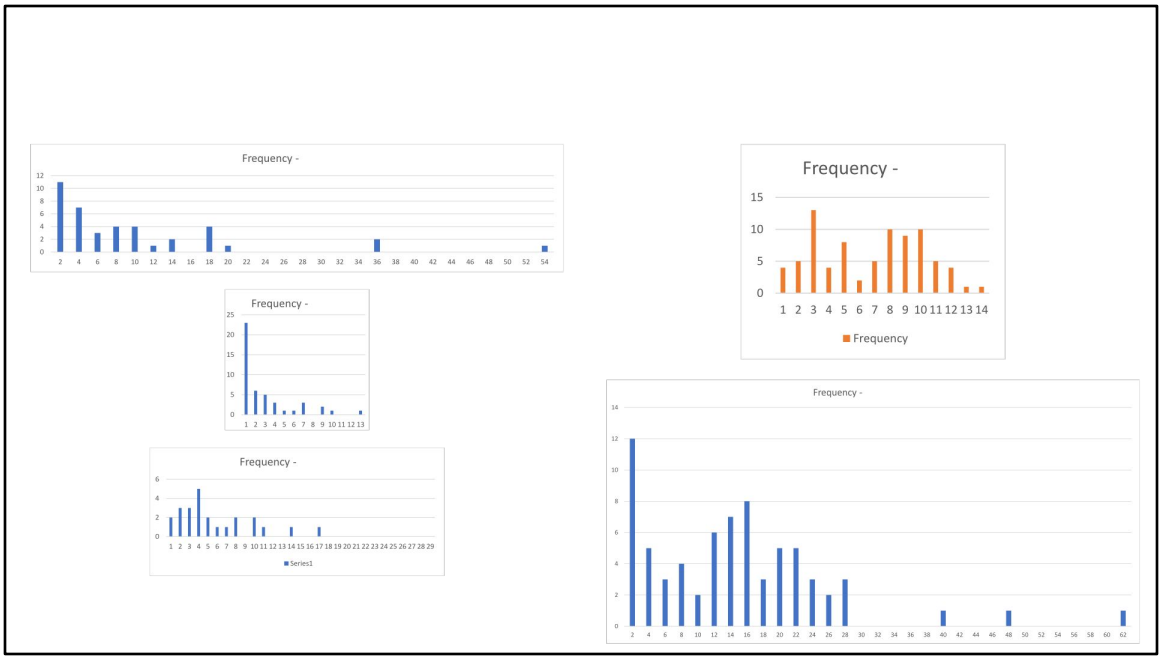








It is the same with every team I have looked at over the past six or seven years. The estimates do not work!



The distributions of job durations tell us a bit of why: They do not follow anything close to a normal distribution. Instead they have long tail distributions.

Also note that the distributions are quite different for different teams.

It is pretty clear estimates do not work. We need to do something different!

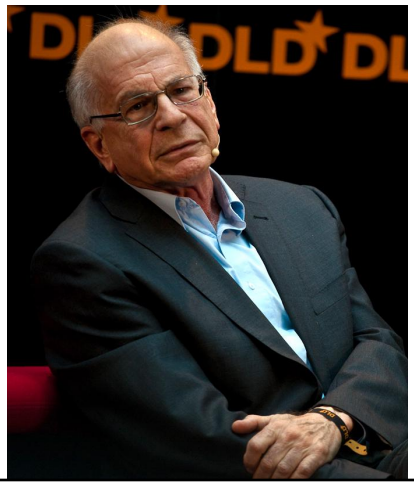
Uniqueness of IT Cost Risk

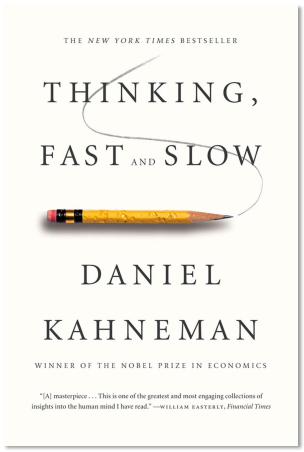
TABLE 4. Cost Risk Categories Based on Alpha-Values (see also Table 3)

Extreme (α ≤ 1)	Very high (1 < α ≤ 2)	High (2 < α ≤ 3)	Elevated (3 < α ≤ 4)	Moderate (α > 4)
• IT	• Buildings • Dams (other) • Defense • Fossil thermal power • Hydroelectric dams • Nuclear power • Nuclear storage • Olympics • Rail	• Aerospace • Bridges • Mining • Pipelines • Rail stations • Roads • Tunnels • Water	• Oil and gas	• Energy transmission • Nuclear decommissioning • Solar power • Wind power

I am not the only one pointing out this problem with estimates. In a 2025 research paper, Uniqueness of IT Cost Risk, the researcher Bent Flyvbjerg, and some of his colleagues, show that IT projects are more unpredictable than other kinds of projects.

Uniqueness of IT Cost Risk





Flyvbjerg's research was inspired by ideas by the Nobel Prize laureate Daniel Kahneman.

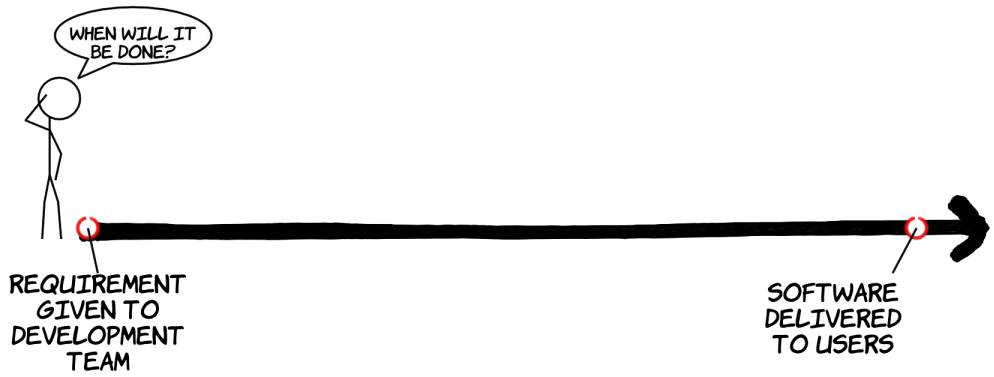
Kahneman noticed that he, and his colleagues never got it quite right when they estimated their projects.

He figured out that the reason was that there are simply to many unknown factors in a complex project. Not only that, the longer a project lasts, the greater the risk that there is some unforeseen delay that wrecks the estimates.

Kahneman also figured out a solution. Flyvbjerg and his research team developed it further, and used it in real, very complex projects.

It is that solution we will explore.

Part 3: A Better Solution!

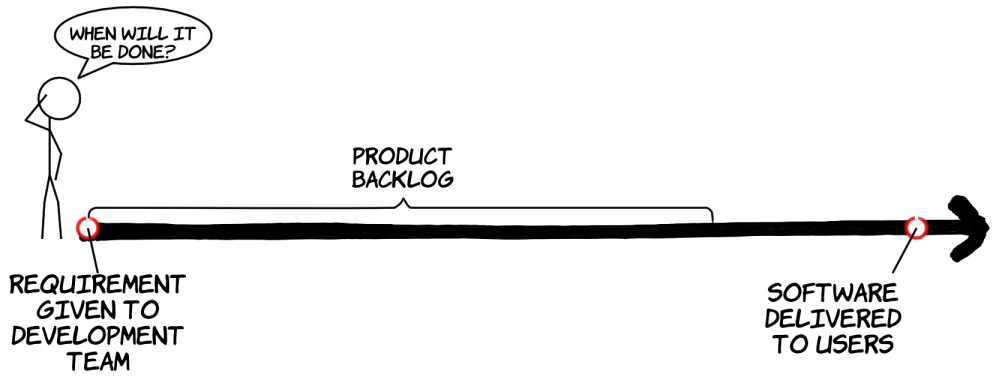


First, let's specify what we mean when we ask for a time estimate.

For a customer, a manager outside the team, or a Product Owner, the most interesting thing to know, is the lead time, that is how long it takes to implement an idea, from when we first give a requirement to a development team, until users get finished functionality.

Let's break the timeline from start to finish down a bit.

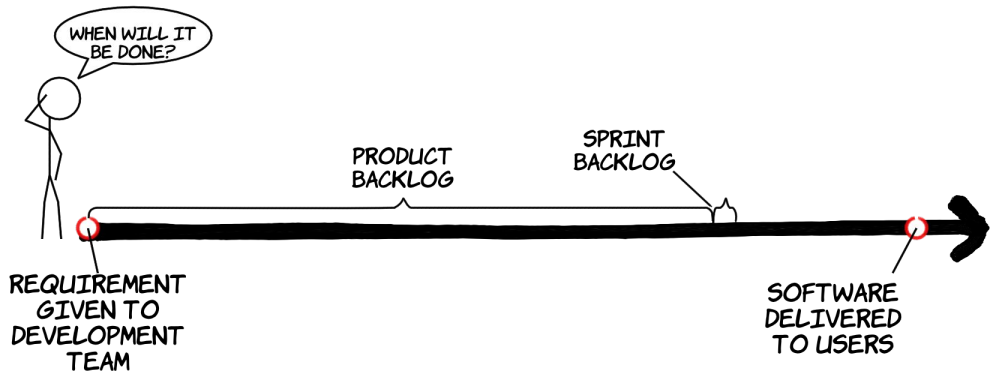
Part 3: A Better Solution!



If we have a Scrum team, the first thing that happens is the requirement is written down, often as a User Story, and put in a Product Backlog.

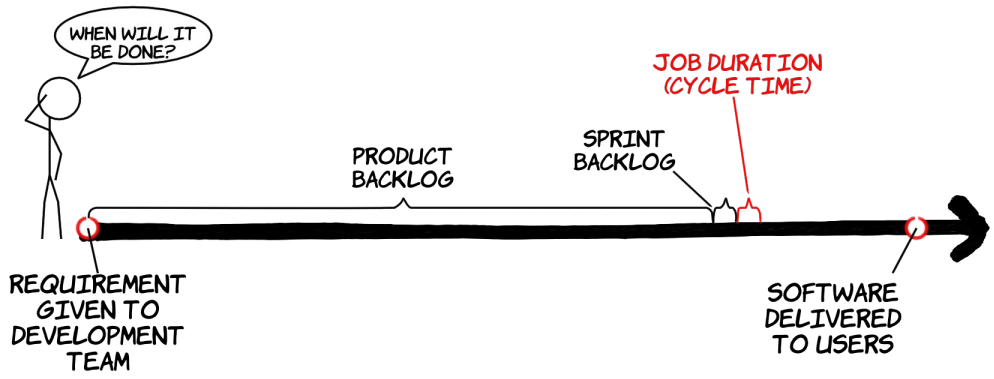
There it lies waiting, until the team has time to do something with it.

Part 3: A Better Solution!



Eventually, the team has a look at the requirement, refines it a bit, and sticks it into the Sprint Backlog.

Part 3: A Better Solution!

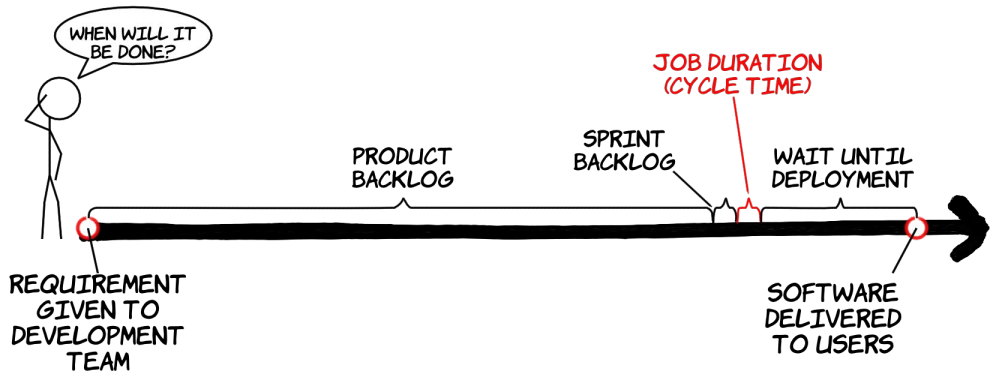


The requirement lies waiting a relatively short time in the sprint backlog. Then, the team works on it.

We often measure the cycle time as the duration from when the team begins to work on a requrement, until the team is done.

It is worth noting that we can measure cycle time from any point in the process, to any other point, so it is a good thing to specify which cycle time we are talking about.

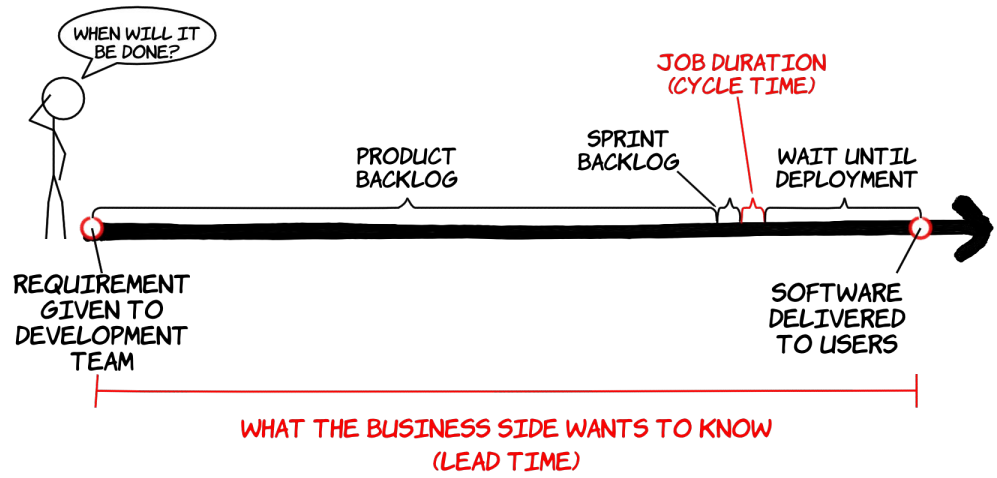
Part 3: A Better Solution!



After the team is done, there is a wait until the software functionality is deployed.

If the team has Continuous Deployment, and it works, this time may be down to a few minutes, but it may be considerably longer.

Part 3: A Better Solution!

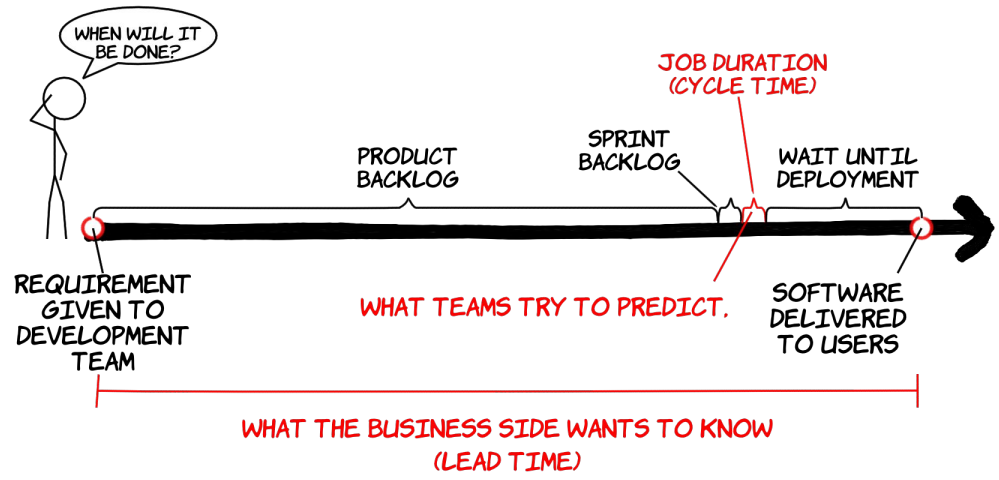


Summing it up, in most cases, this is what the business side wants to know.

The unit of work business is interested in, may be the lead time of User Stories, Features, Capabilities, Epics, Use Cases, an Entire Story Map, a subproject, or a project.

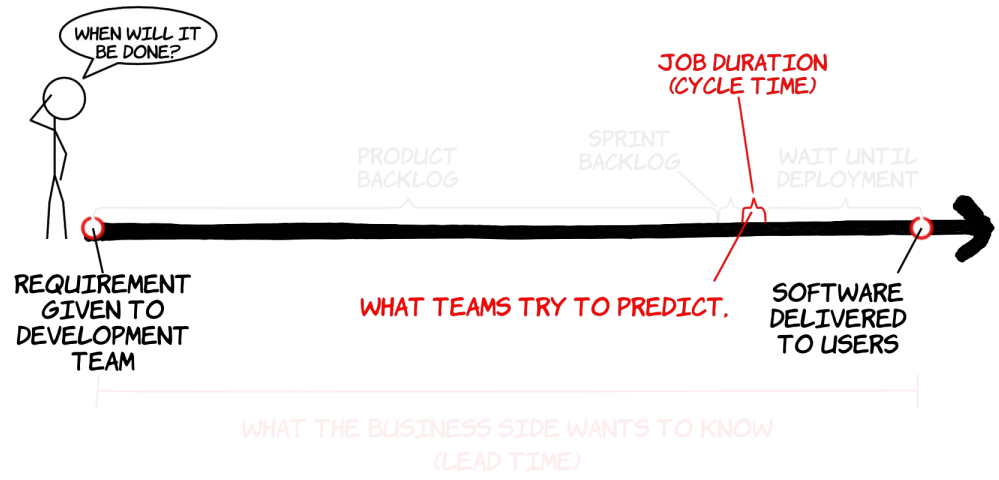
Programs, of course, do not have a defined end state, so the duration of a program cannot be predicted. Instead, a program is run until it is no longer profitable.

Part 3: A Better Solution!



Scrum teams focus on the cycle time, from when they begin to work, until they are done.

Part 3: A Better Solution!

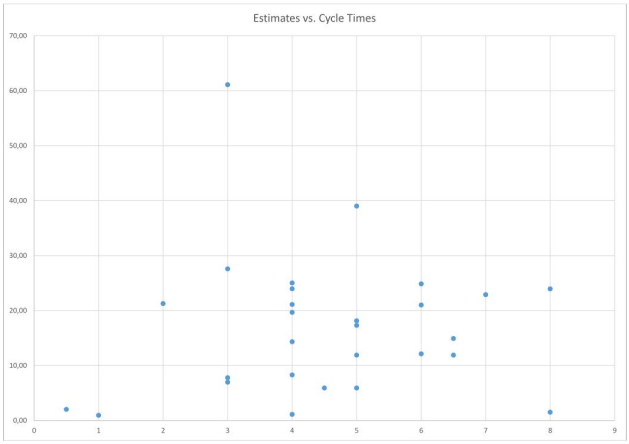


We will have a look at this first.

I am pretty sure many of you will think, “but we estimate effort, not time”.

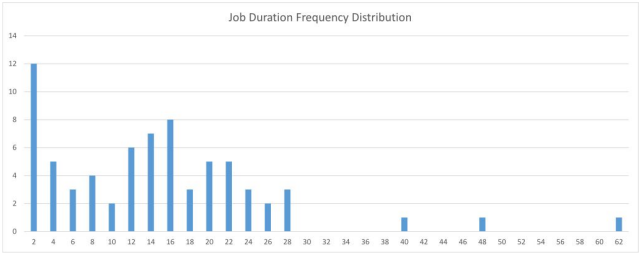
Well, the team does capacity planning for a timeboxed sprint. No matter what you call it, what you do, is trying to figure out how much work you can fit into the timebox.

Part 3: A Better Solution!



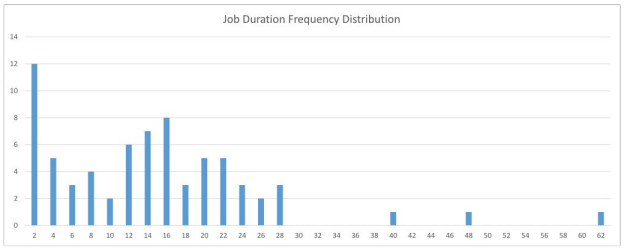
As we have already seen, this is like trying to predict the outcome of a dice roll.

Part 3: A Better Solution!



To make it worse, the dice is uneven, so it has a wonky probability distribution.

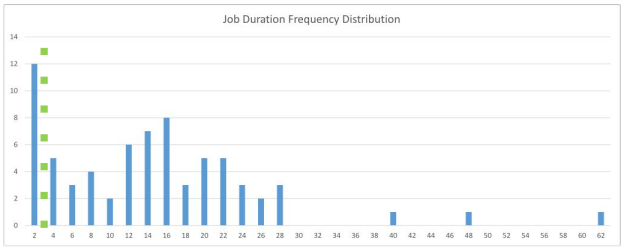
Part 3: A Better Solution!



So, why is this such a problem?

Let's make a thought experiment. We have got a User Story, and we want to make an estimate.

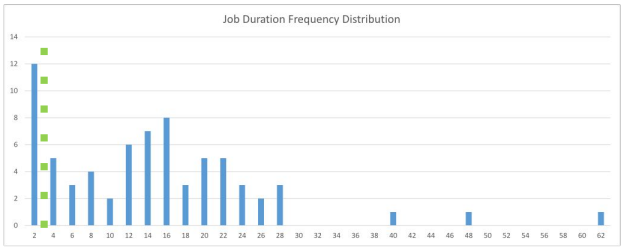
Part 3: A Better Solution!



What is the best estimate?

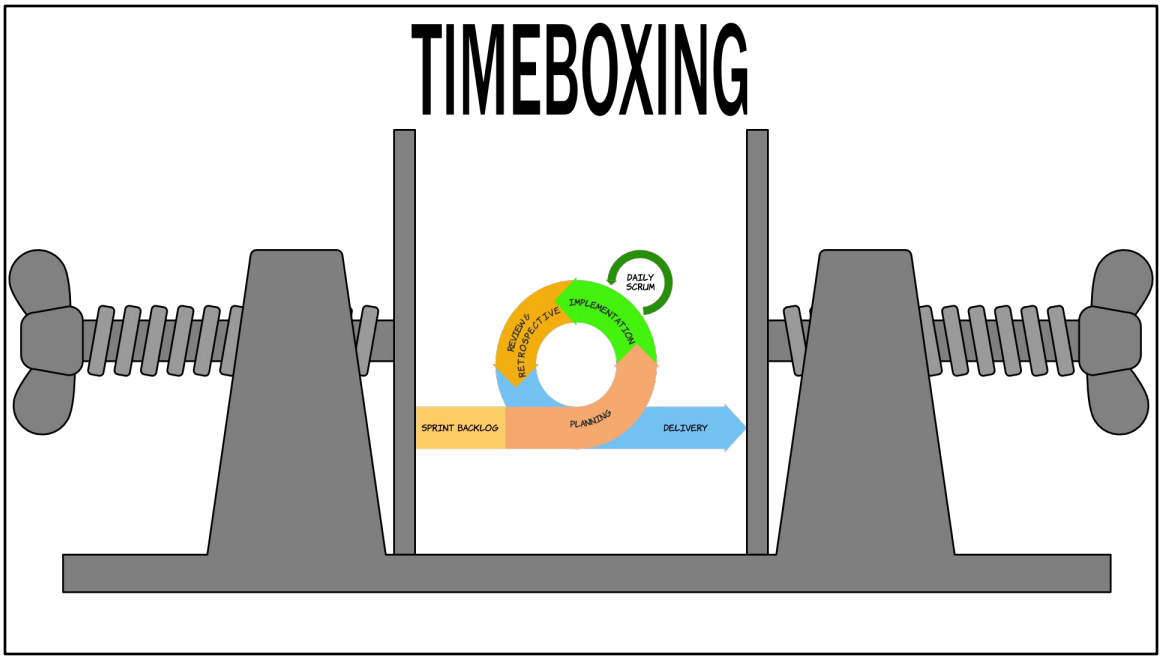
There is no single place that looks like the best guess.

Part 3: A Better Solution!

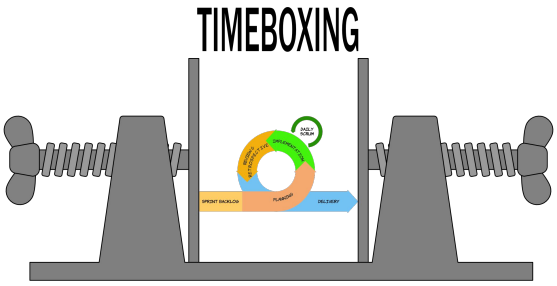


The reality is, the actual duration can end up anywhere within the probability distribution.

There is no single number that represents a good guess.



This has consequences when we use estimates for timeboxing, as in Scrum's Sprint Planning.



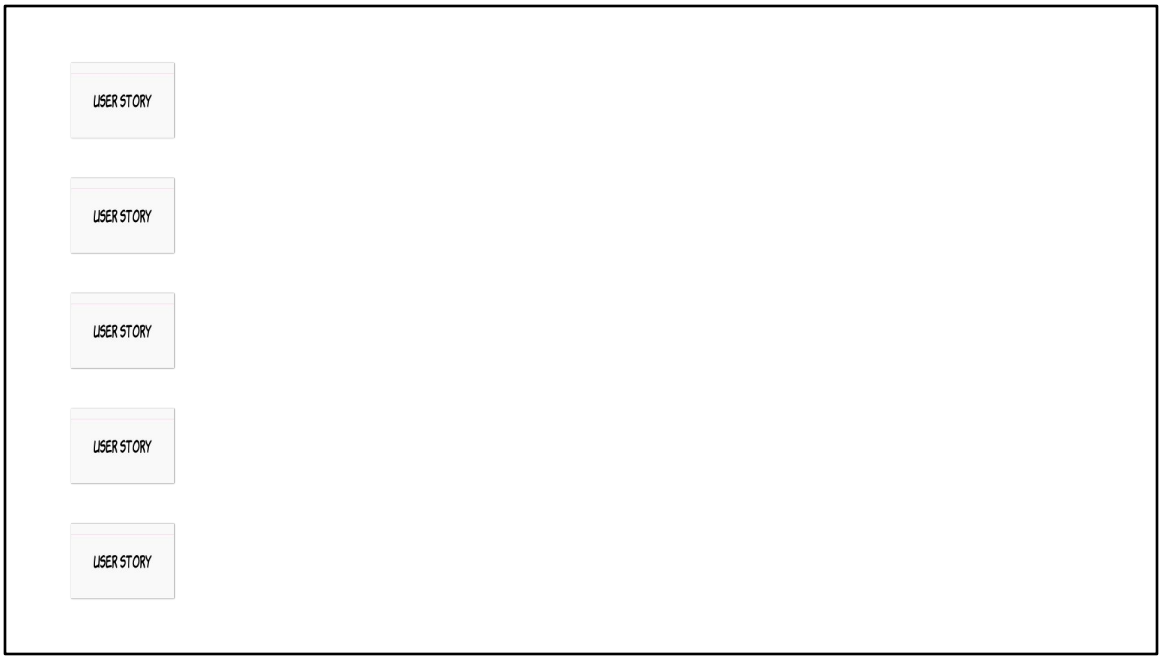
- Fight procrastination
- Focus on work that matters
- Achieve something complete and meaningful

If we look into why we do timeboxing, there are a couple of common reasons.

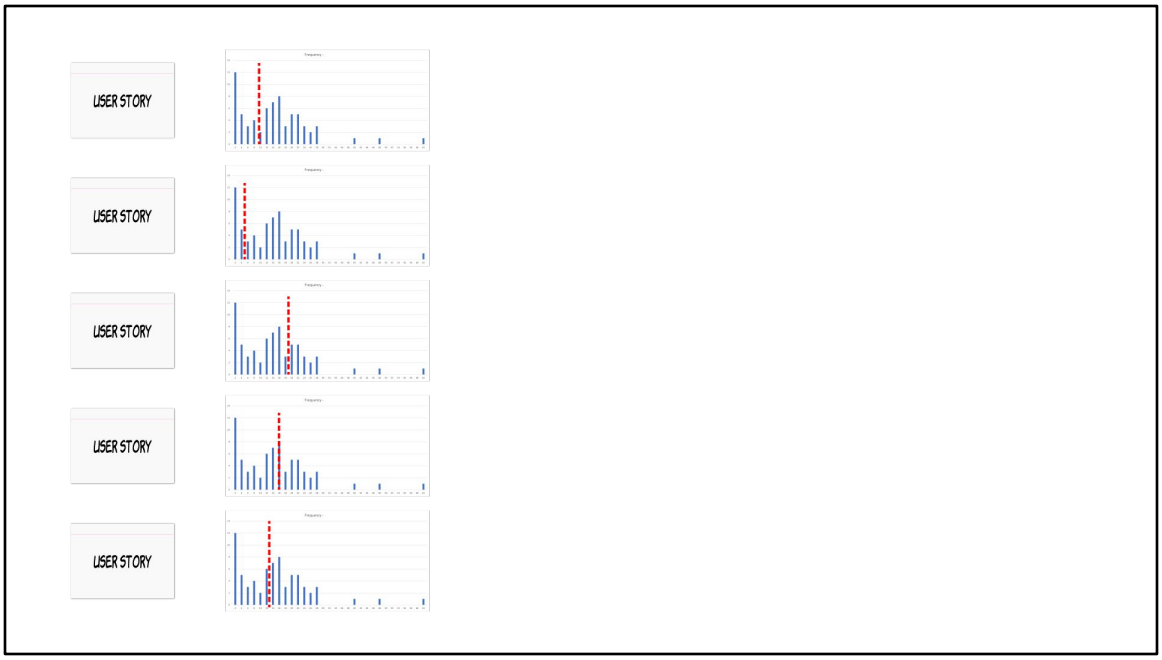
[**CLICK and read each alternative**]

The problem is that if the estimates do not work, then timeboxing will not work.

Let’s have a closer look.

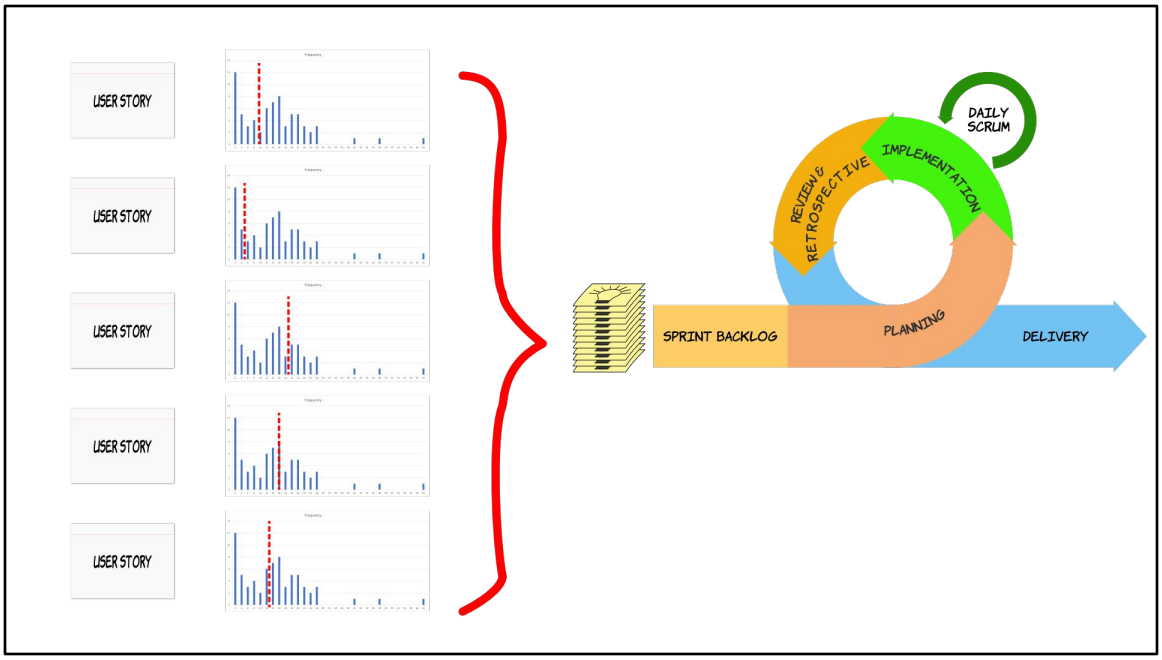


Here are five user stories. Let's estimate them.

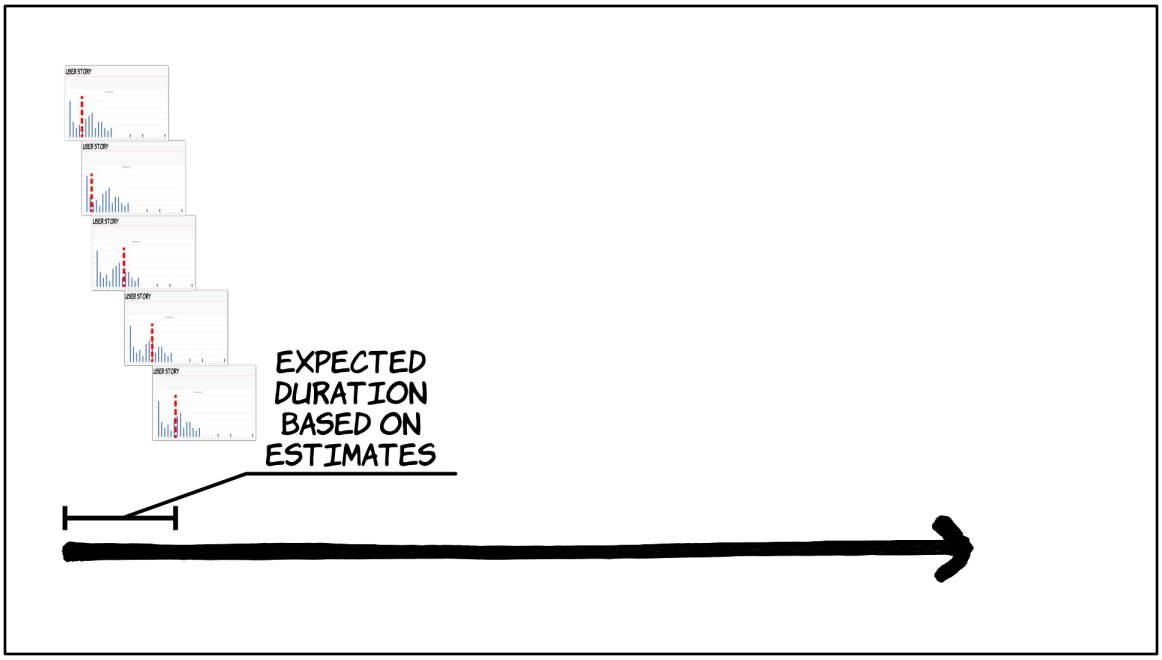


Each User Story has a duration within the probability distribution.

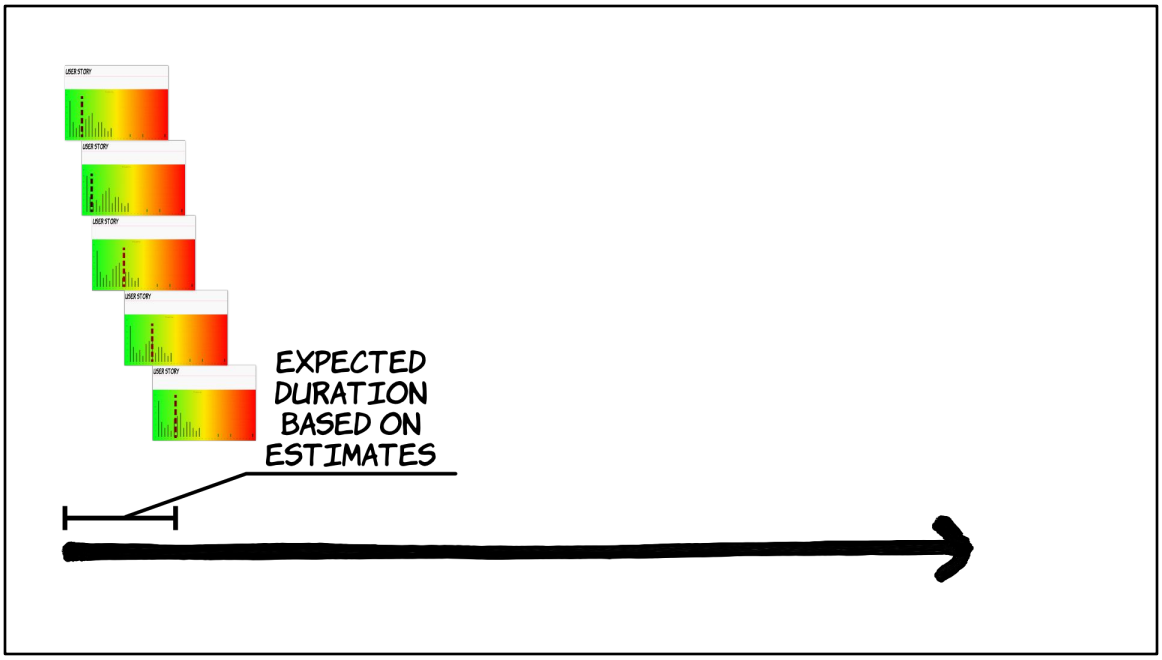
We make the best estimates we can.



Based on the estimates, we determine we should be able to do these five User Stories in a Sprint.



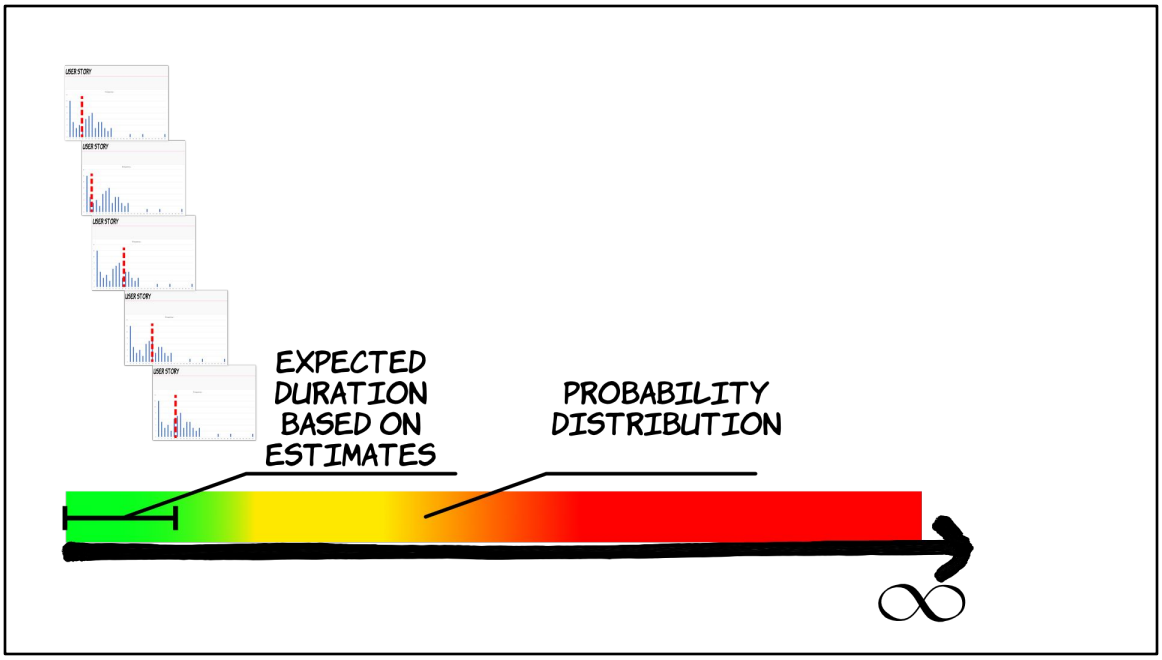
If we sum up our estimates, and show the sum on a timeline, it looks like this.



However, any one of the User Stories has a fairly high probability of taking a lot longer than what we estimated.

There is also a probability that a User Story is finished faster than expected, but the probability distribution is asymmetrical.

Gains tend to be small, while losses can be very large.



If we look at the probability distribution for all of the User Stories together, it is a lot longer than a single Sprint.

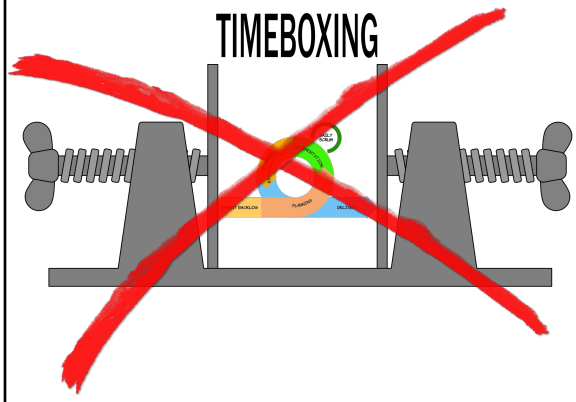
This means we tend to overestimate how much work we will finish in a Sprint.

When that happens, it becomes impossible to finish within the timebox. We fail to meet the expectations, over and over again.

This can turn the timeboxes into torture chambers, and demoralize the best of teams.

[**CLICK!**] Don't freak out! It's still strawberry jam!

Part 3: A Better Solution!



- Fight procrastination
  - Pull processes, pair-programming, troikas
- Focus on work that matters
  - Prioritization, pair-programming, troikas
- Achieve something complete and meaningful
  - Vertically sliced requirements
    - User Stories, Features, Use Cases
      - Thin-slicing
  - Cross-functional teams
  - T-shaped training
    - Train like a Marine!

Therefore, timeboxing is not such a good idea.

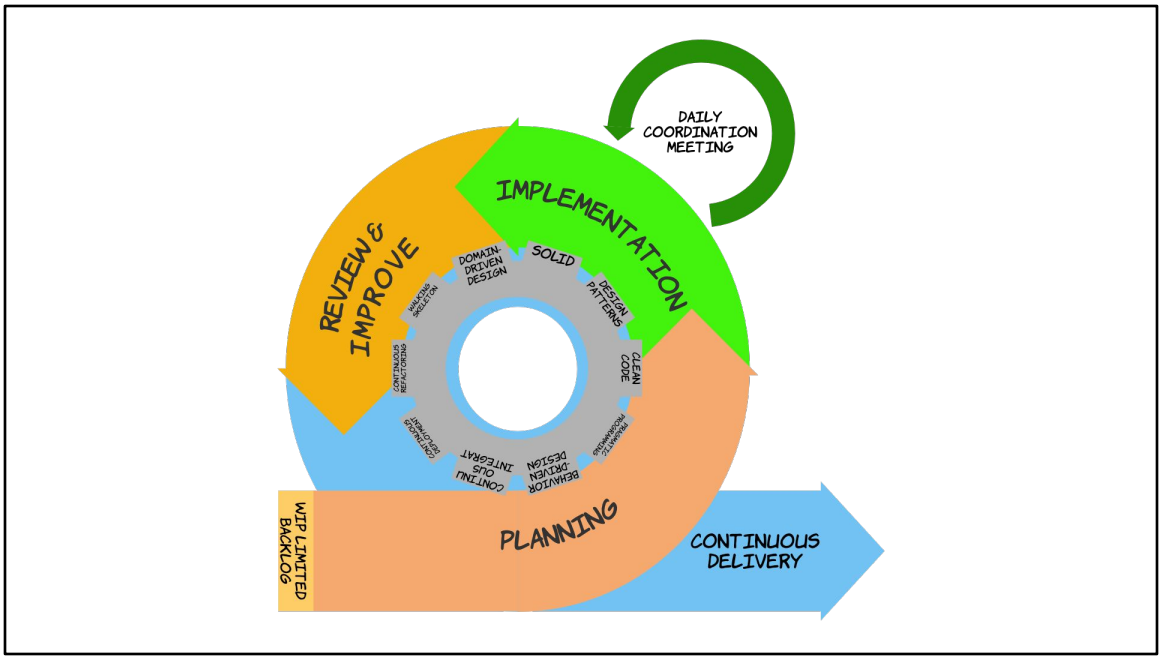
Add to this, that in Agile, we already have practices and techniques that accomplish all of the things timeboxes are supposed to do, but in a much safer manner.

[**CLICK! Click through each item in the bullet list.**]

What all of this means, is that:

Timeboxing does not work

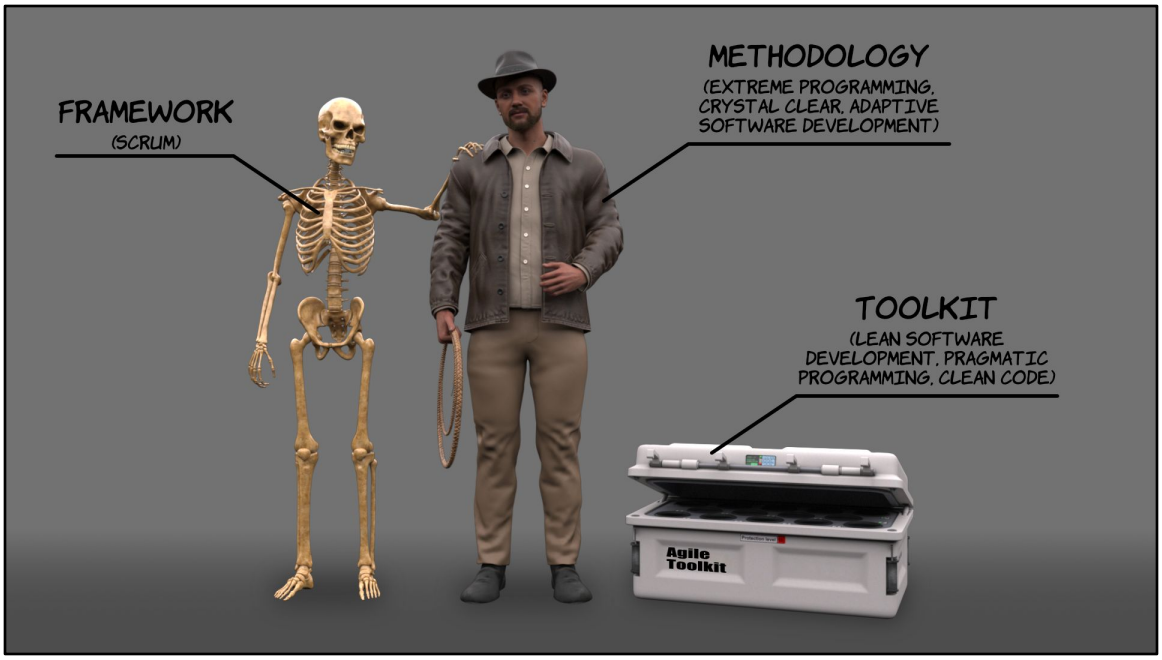
We do not need timeboxing.



A fairly easy way to eliminate timeboxing, is to go fully pull process.

Get rid of the Sprint backlog.
Put a hard WIP limit on the product backlog.
Make a rough prioritization.
Pull the top item off the backlog, refine and implement.
When done, pull the next item off the product backlog.

It could look something like this!



In case the idea of changing Scrum, makes you nervous, I want to remind you that Scrum is a *framework*.

A framework is a supporting structure. You can build a working methodology based on Scrum, but Scrum is not intended to be a methodology all by itself.

A methodology is a system of values and practices that support and reinforce each other. In agile, Extreme Programming is a good example. Methodologies need to be adapted, depending on the context they are used in. Generally speaking, adapting a methodology is *easier* than building a methodology based on a framework.

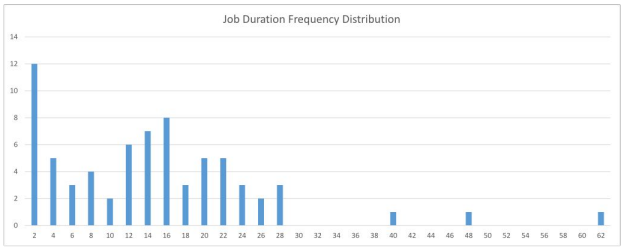
Finally, a toolkit is a collection of methods you can use to flesh out a framework to create a methodology, or, use to adapt an existing methodology so it fits your context.

So, you should not be afraid of changing the Scrum framework.

You *should* be afraid of *not* changing the Scrum framework!

Let's move on!

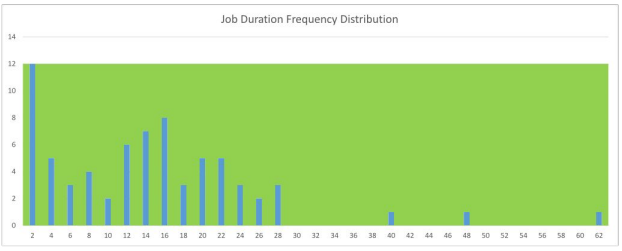
Part 3: A Better Solution!



So, where did we go wrong when trying to figure out when a User Story is done?

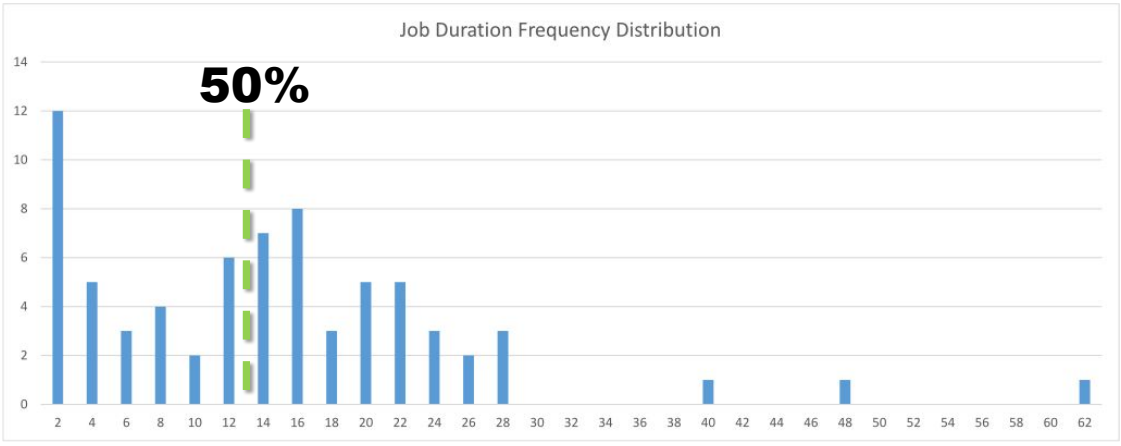
We need to go back to the drawing board, and look at the problem using another perspective.

Part 3: A Better Solution!



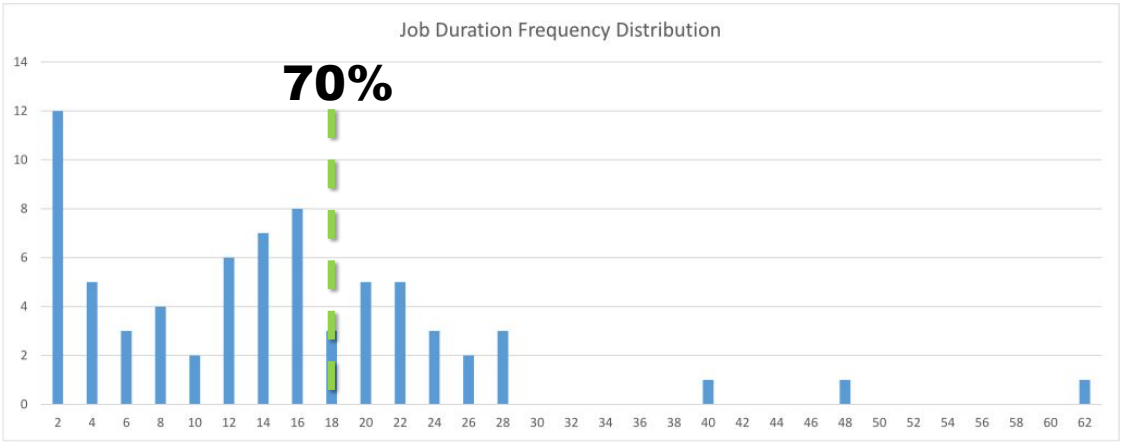
What if we look at the statistical probability distribution, and treat it as, and this may be a shock, a statistical probability distribution?

Part 3: A Better Solution!



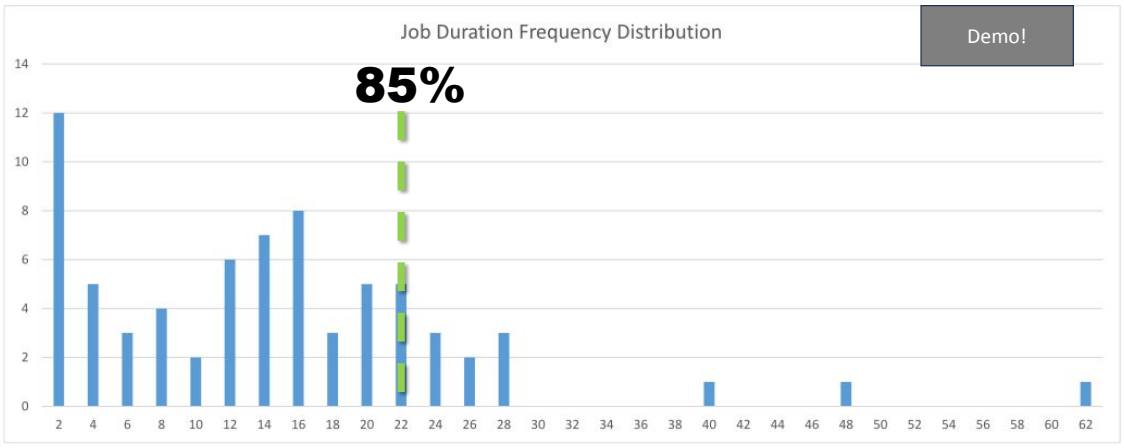
At some point along the X-axis, the duration axis, 50% of all User Stories will be done

Part 3: A Better Solution!



At another point, 70% of all User Stories will be done.

Part 3: A Better Solution!



This is getting a bit repetitive, but there is also a point along the X-axis where 85% of the User Stories will be done. For the team I'm showing data for here, this is 22 days.

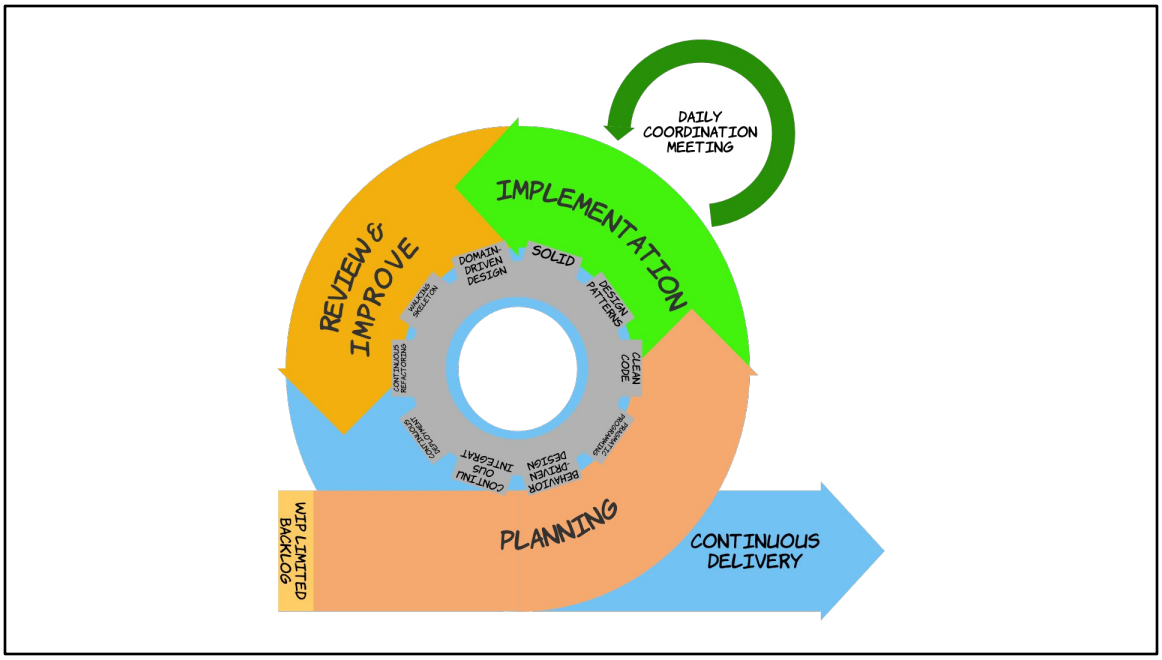
This means, if you pick any User Story at random, we can say it will be done in 22 days, with 85% probability of being correct.

In theory, we could decide what probability we want of being correct, and then use the probability distribution to figure out the duration.

You may ask, how difficult is it to do that? I'm glad you asked.

It turns out we can do that easily, with the *percentile* function in Excel. I'll show you.

[**Click *Demo!* button. Tab *Open Issues 2*. Show how Reference Class Forecasting works.**]



That is pretty much it!

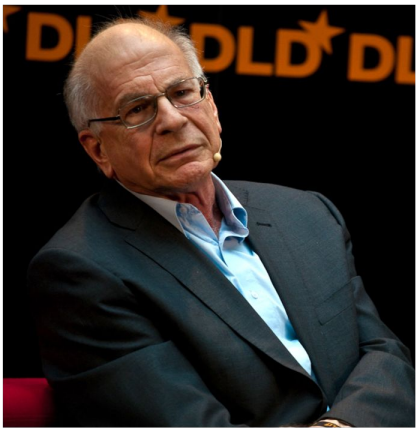
Reference Class Forecasting is completely data-driven, so no one has to do any guesswork whatsoever.

If you have reliable data, the method is easy to implement.

You need to have a stable system, for Reference Class Forecasting to work. To do that, you can use established agile methods for controlling flow and code complexity.

Remember that Scrum is not a methodology! You will need to look at methodologies and toolkits to learn what you need to do, and how to do it. This, however, is nothing new.

Part 3: A Better Solution!



DANIEL KAHNEMAN

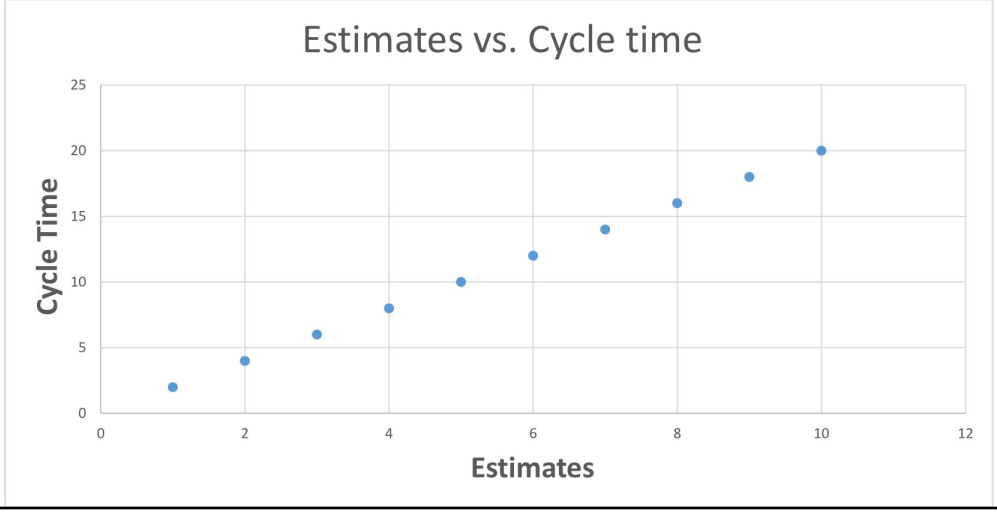
BENT FLYVBJERG

I will advice you to go directly to the sources to learn more.

I first learned about Reference Class Forecasting from Daniel Kahneman, and began figuring out how to use it in practice.

Then, when I found Bent Flyvbjerg's whitepapers, and his book, How Big Things Get Done, I had what I needed to verify that I had understood Kahneman's ideas correctly.

Part 3: A Better Solution!



Don’t forget, that if you switch to Reference Class Forecasting, you need to check that it works for you!

Don’t repeat the mistakes almost everyone does with estimates.

Finally, the struggle for improving how we work goes on!

Never give up!


THANK YOU FOR LISTENING!
Henrik Mårtensson
self@henrikmartensson.org
+46 76 312 70 68