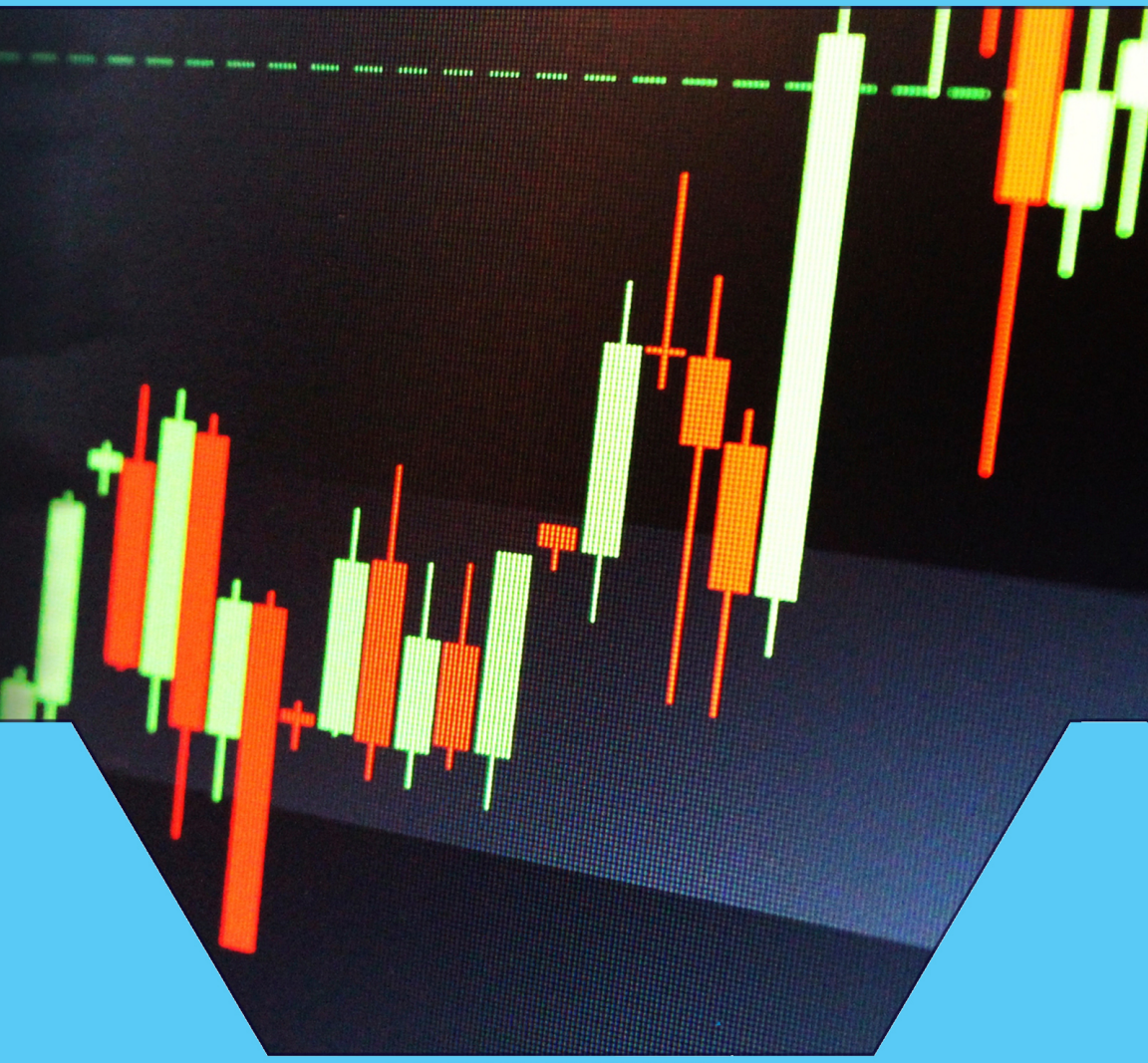
PREDICTING THE UNPREDICTABLE

Pragmatic Approaches to Estimating Project Schedule or Cost



AUTHOR OF *"MANAGE IT!*
YOUR GUIDE TO MODERN, PRAGMATIC PROJECT MANAGEMENT"

JOHANNA ROTHMAN

Predicting the Unpredictable

Pragmatic Approaches to Estimating Project Schedule or Cost

Johanna Rothman

This book is for sale at
http://leanpub.com/predictingtheunpredictable

This version was published on 2015-07-20

ISBN 978-1-943487-03-5

Practical ink

To everyone who was ever asked,
"How much will this project cost?" or
"How long will this project take?"

Contents

1. Introduction

One of the big questions in organizations is "How much will this project cost?" The other question is "When will this project be done?" In fact, the bigger the project, the more the people in the organization want to know.

The problem with these questions is that they are predictions. I don't know about you, but my crystal ball is opaque. It's never been good at predictions.

On the other hand, I've had pretty good results with educated guesses. I've had even better results with using data to update my guesses as I've proceeded.

Estimation has been a common "problem" in software projects. I have written about it in essays and blog posts for several years. I decided if I created a book, I could share what has worked for me, in real projects and programs. You could apply these ideas in your projects and programs.

1.1 Estimates Are Guesses or Predictions

We want accurate estimates. When I drive to a local appointment longer than 10 minutes away, I want to know how long it will take me to get there. I can use a mapping application to estimate my drive, given the current traffic.

I have driven to appointments only 14 miles away, and

sometimes it takes me 20 minutes to drive. Sometimes, it takes me about an hour. That's a huge difference.

My general estimate is that I should give myself 20-25 minutes of driving time. But if I always planned on 25 minutes, I would be late—substantially late—about half the time.

We don't want that with our project estimates. And yet, we are "late" with projects.

When I drive to an appointment, I don't add features. We often add or change requirements with projects and expect the estimate to be the same as predicted.

I don't change who is driving—it's always me. Yet, we change people on teams and expect the estimate to be the same.

I don't multitask when I drive. Yet, we ask people to multitask and expect the estimate to be the same.

An initial project estimate is your best guess at the time you make the estimate. The accuracy of that estimate depends on the people doing the estimation and what they know about the project.

Your estimate is a guess, a prediction. It is not fact. If I have trouble estimating my driving time, which has far fewer variabilities than a project, how can you expect your project estimate to be accurate? You can't, not with the very first estimate you create. You can iterate on the estimate and make it better over time.

1.2 Estimates Change

As you proceed with your project, your estimate will change. Your estimate might change because:

- You know more about the project's features/requirements.
- The people understand how to work together.
- You have history with these people working on this project. That history will allow you to better your prediction.
- The project itself changes: adding, subtracting, and changing requirements as we learn more about what we can provide and more about what the customer desires.

Note that in my driving metaphor, my destination doesn't change—unless I abandon my drive! Because projects change, our first estimate is a guess or prediction. That's why it's so important to update your estimate as you proceed.

1.3 Estimates Expire

Not only can the project itself change, which changes the estimate, but estimates have an expiration date.

On an agile project, as you finish features, you learn more about the requirements. You learn what a Minimum Viable Product, MVP, is. You might need a Minimum Indispensable Feature Set first, enabling you to create short features in flow or iterations.

If you work in a non-agile way, you will address risks—technical, schedule, or quality risks—and learn from them.

Regardless of your type of project, the estimate you create at the very beginning will expire. It is no longer valid.

Keep these ideas in mind as you read this book. These ideas are true regardless of your project's life cycle.

I've collected here what I've written over the years to provide guidance about estimation. I hope you use it and enjoy it.

2. What Estimates Are

Managers and sponsors ask a project team for estimates all the time. Sometimes, managers use those estimates to plan which project to do first—which is a terrible idea. More on that later. Sometimes, managers use those estimates to predict a target date. Sometimes, managers use those estimates as a commitment.

None of those ways are the way we should use estimates.

Here is the problem with using estimates that way:

- Estimates expire.
- Estimates change.
- Estimates are guesses.

Yes, "guess" is the dictionary definition of estimate. The more you know about the domain, the requirements, and how the team works together, the more educated your guess will be. But it will still be a guess.

Because estimates are guesses, you can't use them for precise prediction until you get close to the event you are estimating.

You can spiral in on more accurate estimates. You can provide a percentage confidence on your estimate as you proceed through your work. You can change your estimate as you learn more about the necessary (and unnecessary) requirements.

2.1 Provide an Accurate but Not Precise Estimate

Can you have an accurate estimate at the beginning of a project? It depends. If the project is similar to what you have done before, with a team that is accustomed to working together, and the effort is short—say no longer than three months—you have a good chance of providing an accurate estimate.

For our purposes, an accurate estimate is one that is no more or less than 5-10% off the final effort.

If you have a 12-week effort, and you finished inside of a week either way, would you consider that a good estimate? I would. That's what I mean by an accurate estimate.

Instead, if you say, "We will finish on Wednesday, the 23rd at 5pm," would anyone believe you? They might, if you have an important demo or release scheduled for Wednesday, the 23rd at 6pm! That's a precise estimate.

Maybe, you use that precise date as a target, a predictive estimate. You timebox the work so you can finish it by that time. That's useful to do with trade shows, demos, and other events where you are not in charge of the date.

When someone asks you, "How long will the project take?" or "How much will the project cost?" they are not talking about a precise estimate. They want an accurate estimate. You need to know the difference.

To manage people's expectations, always report your estimate with a percentage confidence, a date range (optimistic, likely, pessimistic), or spiral in on a date.

Now that you know about the fallibility of estimates, let's consider why we estimate.

3. Why Do We Estimate Anyway?

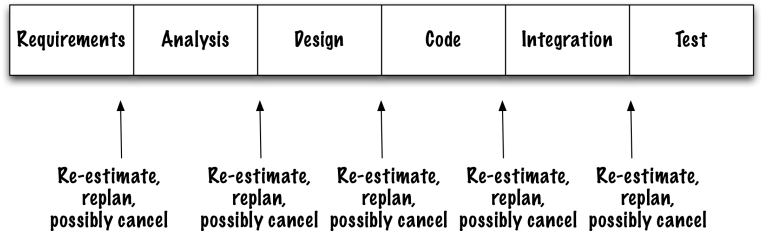
We estimate for these reasons:

- To provide an order-of-magnitude size/cost/date about the project, so we have a rough idea of the size/cost/-date for planning purposes. An order-of-magnitude size means we want to invest just enough time in the estimate that we believe in the accuracy of it for planning purposes.
- We want to know when we will be done, because we are close.
- We need to allocate money or teams of people for some amount of time.
- Someone wants to know who to blame.

Creating a gross estimate can be useful. See Estimating the Unknown: Dates or Budgets to learn how. You can iterate on that estimate, especially as you get closer to completion.

A gross estimate with deliverable milestones can help the organization allocate money or teams for a while. The deliverable milestones help you know when you have completed enough value. The value is a compilation of features, something you can demo.

In fact, back when phase-gate (serial) life cycles were developed, estimation was key to a project's success.



Role of Estimation in a Serial Life Cycle

After you completed a phase, you were supposed to re-estimate, in order to change what you did with the project.

Now, if you are using an incremental approach or an agile approach, you see completed work. And you don't have to re-estimate. However, you should be aware that many managers may be thinking about a serial life cycle when they ask for estimates.

3.1 Why Do You Estimate?

Why do *you* estimate? If you've estimated because you always have, think about it. If you estimate because your money people want to do once-a-year money allocation, well, you know that's fiction. You can do it without detailed project estimation.

For money allocation, decide how valuable the project is to you. When does the project have to deliver the value? Now, tell the project team when the value has to be delivered. That's all.

Remember, you hired these people because they were smart, responsible human beings. Stop with the phases and all that nonsense. Tell them what you want. Remember, the phases

exist because management wanted to be able to cancel the project before it got too far along. You were supposed to show a deliverable and re-estimate at each phase. If you don't cancel or deliver something and re-estimate at each phase, your phases are not working for you.

Buy your team a copy of *Manage It! Your Guide to Modern, Pragmatic Project Management*, (ROTPM), which explains how to manage projects in any life cycle. Give them a ranked backlog. Let them deliver. If they can't deliver in the money or date frame, they will tell you. They are responsible humans.

If you need an order-of-magnitude estimation, fine. That doesn't take days to determine. That takes hours. It will be precise-wrong and order-of-magnitude-right. Timebox your estimation effort. It's an order of magnitude. Don't hold anyone to that estimate. (Remember, estimates are guesses. They are not "The One and Only Truth.")

If you want to know when you'll be done because you think you're close to the end of the project, ask yourself this question: Is it worth the time to estimate versus the time to finish? It might be. But know you are taking time away from finishing.

And if you want to play the blame game, remember that management is the one who needs to shoulder the most blame. Why? Because management set the constraints. Don't believe me? Read Estimating the Unknown: Dates or Budgets now.

I can sympathize with management's need for estimates. I like order-of-magnitude estimates for many things. I even like specific estimates as we get closer. But creating software is not like driving somewhere or like constructing a building. When I drive somewhere, I do want step-by-step instructions. When

constructing a building, I do want an estimate. And even then, I am pretty sure the estimate is optimistic.

When creating software, I want to see working software as we create it, because with working software, we learn. The learning is what's most important. Because once we've learned enough, we can stop. That's what's most valuable. Not the estimate.

3.2 Ask This Question Before You Estimate

When people ask you for your estimates, they are trying to determine the value of the project to the organization.

You can ask them to articulate their desires with either of these questions:

- How much do you want to invest before we stop?
- How much value is this project or program worth to you?

Start a conversation with your sponsor, so you can understand what is important to your sponsor. Once you do, you can decide what to do next. You may want a gross estimate, as in Estimating the Unknown: Dates or Budgets. You may want to change your project approach, and do some up-front work to generate a more detailed estimate. You have choices.

4. Software is Learning, Not Construction

Here's one problem I have with estimation. Software is not construction. We can't build software the same way we construct or manufacture something. Software is all about learning and innovating as a team. Some people think that software is invention. Whatever you think about software, it is *not* construction.

We can timebox our learning. We can choose to stop doing something. We can put acceptance or release criteria around it and say, "We have done enough for now."

But, we cannot say, "We can build this software for $xx per square foot." We don't know how to do that. Because we have not built exactly this software before. If we had built software like this before, we could estimate pretty darn close, because we either have historical data with good estimation quality, or we have small chunks of work we know about, or both.

(There are some tools, such as Cocomo, SLIM, and function point counting that ask you many questions at the start of your project to provide an estimate for the entire project. If you are not agile, you may want to consider those tools. Know that you will invest substantial time preparing for the estimate, time that takes you away from helping the team learn what to do or how to work together. The people who sell these tools and services are convinced they work. I am not.)

When we estimate, other people think of our estimates the way they think of estimates in other fields, especially construction. Especially if you provide a single-point estimate. Even if you provide assumptions, which no one hears.

Software is nothing like construction. Software is innovation. Innovation is difficult—if not impossible—to predict.

Since software is about learning, and we rarely, if ever, do the same project twice, we are always estimating the unknown. That makes our estimates inaccurate.

There is an alternative to estimating.

Make your features small, as in something you can deliver in a day or so. You can also swarm over the work, so the team finishes a story every day. If you finish something each day, people can see your work product. They trust you. They stop asking you for estimates.

If you always have deliverable software—this includes all tests, documentation, everything you need—you don't need to estimate anything. You also gain the benefit of learning, so if someone asks, "How hard is this thing to do?" the entire team can huddle together for a few minutes and say, "It's this story and that story and this story, too."

They then say, "We know it's at least these three stories, and that's off the top of our heads. Are those stories more important than the ones at the top of our queue?"

4.1 Inch-Pebbles or Small Stories Show Progress

Because we learn when we write software, we need to show progress to ourselves and to our customers. Small features or small tasks show progress and build trust. It's also much easier to estimate something small than it is something large.

If you create small stories or inch-pebbles, you can track how long it took you to create those stories. Or, you can ask, "Is this story similar in size to that one?"

You learn about how large one or two days of work is. You want to know this. The smaller the granularity of stories or work, the easier it is to count them. You will have a more accurate estimate.

4.2 Learn With Spikes

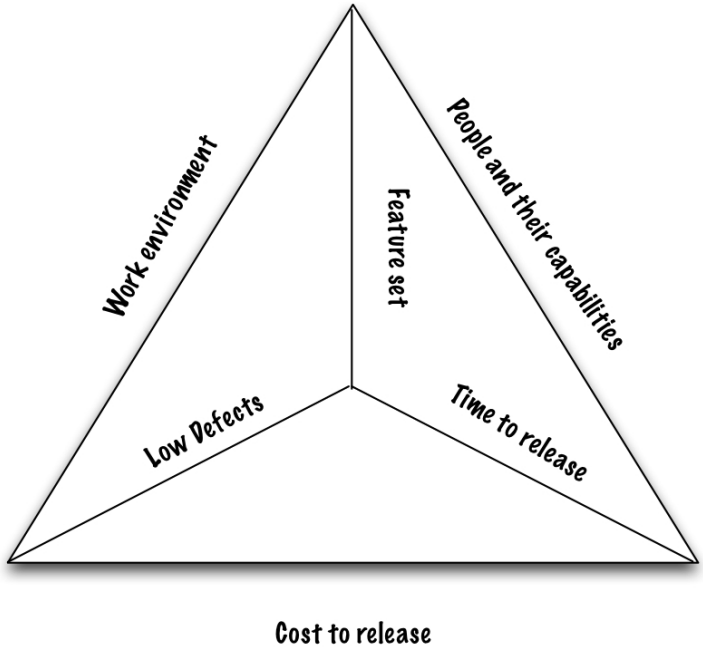
Sometimes you have something you don't even know how to start estimating. You don't even know how to start the work. That's when you need a spike. See Spike It!.

You timebox a short amount of time—say a day or less—and work with your team to determine what the next steps are. At the end of this timebox, you and the team probably know enough about the work to break it down to estimate it.

Take every opportunity to learn as you estimate. You will have a much more accurate estimate.

5. Think About Estimation

First, remember that a project is a system. And, a system has multiple aspects.



Project Pyramid

If you've been managing projects for a while, you know that there is no iron triangle. Instead, there is a project pyramid.

On the outside, there are the typical corporate constraints: Who will work on the project (the people and their capabilities), the work environment, and the cost to release. Most often, those are fixed by the organization. "Bud, we'll give you 50 people, 5 months, and this pile of money to go do that project. OK?"

The inside edges are what the customer wants: A specific feature set, at a reasonable level of quality by a certain date.

Whether or not it's okay, you're supposed to nod your head like a bobble-headed doll. But, if your management has not thought about the constraints, they may be asking you to smush more features than your people can accomplish in the allotted time, given the requested time to release, with the expected number of low defects and the expected cost to release.

The time to release is dependent on the number of people and their capabilities as well as the project environment. You can *make* anything work. You may have delays with geographically distributed teams, and even life cycles that do not include iteration with long lists of features.

This is why estimation of the budget or the time to release is so difficult. You have to consider the entire context.

5.1 Estimating the Unknown: Dates or Budgets

Almost every manager I know wants to know when a project will be done. Some managers decree when a project will be done. Some managers think they can decree both the date and

the feature set. There is one other tiny small subset, those managers who ask, "When can you finish this set of ranked features?"

And, some managers want you to estimate the budget as well as the date. And now, you're off into la-la land. Look, if you had any predictive power, you'd be off somewhere gambling, making a ton of money. But, you do have options. All of them require iterating on the estimates and the project.

First, a couple of cautions:

1. Never, ever, ever provide a single date for a project or a single point for a budget without a range or a confidence level.
2. Expect to iterate on the release date and on the budget, and train your managers to expect iterative, improved estimates from you.
3. If you get a ranked feature set, you can provide working product in the order in which your managers want the work done, while you keep refining your estimates. This has to be good for everyone.
4. If you can say this without being patronizing, practice saying, "Remember, the definition of estimate is guess."

So now that you know why it's so difficult to estimate, what do you do when someone asks you for an estimate?

5.2 Determine Your Degrees of Freedom

First, you ask a question back: "What's most important to you? Imagine it's three weeks before our desired release date.

We don't have all the features done. We have more defects than we wanted. What do you want to do?"

- If they say, "Features. Finish the features," then you need to optimize for finishing features. You have to manage technical risk. Features are driving your project.
- If they say, "Date. If we don't make the date, we are toast," then you know you have to timebox everything so you meet the date with completed work. The release date is driving the project.
- If they say, "Cost," then you know to manage the run rate. Make sure people aren't multitasking. Cost-to-release is the driver.
- If they say, "Low defects," then you know you need the team to complete one feature at a time before they start anything else. Low defects is the driver.

If you review the Project Pyramid, you can see that everything depends on everything else. However, in each project or program, you only have one #1 priority. That priority is either the release date, the feature set, the cost-to-release, or a low defect count. Your management might want to contain costs or use certain people, or somehow change or orient the environment some way, but that's not what you deliver to your customers.

You deliver a set of features, on a release date, with a certain level of defects, for a certain cost. One of those is your driver. The rest are constraints or floats of some sort. See *Manage It! Your Guide to Modern, Project Management*, (ROTPM) for more information about how to determine your drivers, constraints, and floats.

Remember, as you consider what's driving your project, you can have only one #1 priority. You might have a right-behind-it #2 priority, and a right-behind-that #3 priority, but you need to know where your constraints and degrees of freedom are.

This is your chance to rank each of the vectors in the pyramid. If feature set is first, fine. If time-to-release is first, fine, if cost is first, fine. If low defects is first, fine. Whatever is first, you don't really care, *as long as you know* and *as long as you only have one #1 priority*. You run into trouble on estimates when your management wants to fix two out of the six sides of the pyramid–or worse–more than two sides.

When your managers say to you, "Here's the team, here's the office space, here's the budget, here's the feature set, and here's the time," you only have defects left to negotiate. And, we all know what happens. The defects go sky high, and you also de-scope at the end of the project because you run out of time. That's because you have too many fixed constraints.

5.3 Insist on a Ranked Backlog

If you really want to estimate a date or a budget, here is how to do it. Make sure you meet these conditions:

1. You must have a ranked backlog. You don't need a final backlog. You can certainly accommodate a changing backlog. But you need a ranked backlog. This way, if the backlog changes, you know that you and the team are working on the work in the correct order.
2. The team who will do the work is the team who is doing all the estimation. Only the team who is doing the work

can estimate the work. Otherwise the estimate is not useful. Surrogate estimators are biased estimators.

3. You report all estimates with a confidence range. If you report estimates as a single point in time, people think your estimates are accurate and precise. If you report them as a confidence range, people realize how inaccurate and imprecise your estimates are, and you have a shot of people treating them as real estimates.

You need a ranked backlog because the order of the features matters. Let's use dinner as an example. If you eat dessert before dinner, you might not want dinner. Why bother estimating how long it will take to make dinner if you're not going to eat it? If you provide some features, and the customer says, "Thanks, this is great. You can stop now," you don't need the rest of the features. You don't need to estimate them, either.

Once you've met the conditions, you can estimate. The same reasoning works for both project dates and budgets.

5.4 The Team Doing the Work Provides the Estimate

Once you have a ranked backlog, make sure the team doing the work estimates its own work. Otherwise the estimate is not useful. Surrogate estimators are biased estimators.

Managers and senior architects tend to underestimate the amount of work. They tend to think the work is easy to do, especially if the requirements are clear.

Other teams might overestimate the amount of work. That's because they are not familiar with the domain, the requirements, the code base, or the tests.

You cannot depend on anyone's estimates except for those of the team doing the work.

6. More from Johanna

I consult, speak, and train about all aspects of managing product development. I have a distinctly agile bent. I'm more interested in helping you become more effective than I am in sticking with some specific approach. There's a reason my newsletter is called the "Pragmatic Manager"—that's because I am!

If you liked this book, you might like the other books I've written:

- Agile and Lean Program Management: Scaling Collaboration Across the Organization
- Diving for Hidden Treasures: Discovering the Value in Your Project Portfolio (with Jutta Eckstein)
- Manage Your Job Search
- Hiring Geeks That Fit
- Manage Your Project Portfolio: Increase Your Capacity and Finish More Projects
- Manage It! Your Guide to Modern, Pragmatic Project Management
- Behind Closed Doors: Secrets of Great Management (with Esther Derby)

In addition, I have essays in:

- Readings for Problem-Solving Leadership
- Center Enter Turn Sustain: Essays on Change Artistry

I'd like to stay in touch with you. If you don't already subscribe, please sign up for my email newsletter, the Pragmatic Manager, on my website. Please do invite me to connect with you on LinkedIn, and follow me on Twitter, @johannarothman.

I would love to know what you think of this book. If you write a review of it somewhere, please let me know. Thanks!