Extracted from:

Software Estimation Without Guessing

Effective Planning in an Imperfect World

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

Software Estimation Without Guessing

Effective Planning in an Imperfect World

George Dinwiddie
edited by Adaobi Obi Tulton

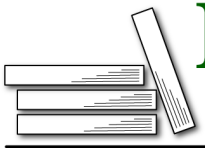
Software Estimation Without Guessing

Effective Planning in an Imperfect World

George Dinwiddie

The Pragmatic Bookshelf

Raleigh, North Carolina



Our Pragmatic books, screencasts, and audio books can help you and your team create better software and have more fun. Visit us at *https://pragprog.com*.

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-698-3
Encoded using the finest acid-free high-entropy binary digits.
Book version: B1.0—September 11, 2019

Introduction

If you've been working professionally in software development for a year or more, I'm sure you've sometimes thought that estimation is a pain in the patootie. It's expensive, contentious, and difficult. Some people consider estimation to be an essential, if distasteful, aspect of software development. Others consider it irrelevant. And I'm sure there are some who estimate out of habit or tradition without giving it much existential consideration. Others argue against estimation to counter such habits.

On the other hand, estimates are all you can know about the future. Within limits, estimates can help you make decisions. You need to be congnizant of the limitations of estimates and the risks of believing that they are data. Armed with those precautions, then you can use them more freely and discover actual data when reality disagrees with them.

Numbers are not the goal. Successful outcomes are. Those footing the bill would rather have a successful outcome than an excuse to blame the estimator. In exploring better estimation, I'm not looking to protect the estimator or development organization from censure or litigation after a disappointment. Instead, use estimation to help guide the project to success from the point of view of both those paying for development and those performing it. There's a lot of benefit to be gained, much more than people usually realize.

Benefit of Headlights

There is a reason to burn headlights when driving at night and to slow down when driving in fog. Headlights allow you to anticipate the need for changes in direction earlier, and therefore achieve faster flow with no loss in caution. In the fog, you need to slow down precisely because you can't see as far. It's helpful to see the shape of the next bend in the road. It's helpful to know approximately how far to our next planned turn or stop. You can benefit from knowing where you might stop for a meal or a night's lodging.

The same is true in software development. You can go faster when you can see further than the work right in front of you. You can strategize how to incorporate the bigger picture. You can avoid painting yourself into a corner far from the door you want to take.

And if you're meeting a hard deadline, you might have better options than developing user stories in prioritized order until you run out of time. In the case where what you want to build won't fit into the time available, you might do better to add some polish on what you deliver rather than maximizing functionality. This will give the user an impression of work that's been finished rather than merely stopped. It's the equivalent of turning down a side road to a comfortable hotel instead of spending the night in the backseat of a car on the shoulder of the road.

If a development team can benefit from seeing a few weeks ahead, you can imagine that the larger organization around that team would like to see even further. Some of this desire may be due to artificial but customary rhythms, such as annual budget cycles. Such things are likely hard to change in corporations and likely impossible in government organizations. Even if you can be successful in changing the rhythm, it will take time to do so. And that is time the organization would like you to use developing systems for use or sale. Beyond that, there are rhythms based on a more substantial basis, such as the amount of time it takes to do related work, the changing of the seasons, or planetary ephemera. And when the CEO asks what's coming up, the CIO would like to have a satisfactory answer.

Beyond Story Points and Planning Poker

If you're unfamiliar with Agile Software Development, the terms "story points" and "planning poker" may be completely mysterious to you. If you're new to Agile, these terms may have specific meaning to you that aren't universally true. These concepts have a history, and are best understood in the context of that history.

Story Points

In the beginning, there was the Chrysler Comprehensive Compensation or C3 project, the birthplace of eXtreme Programming. The short version of the story is that some very smart people tried some things until they got something that worked, and that included breaking the work to be done up into User Stories (c.f., User Stories, on page ?) that could be independently developed and tested. For planning purposes, these were estimated in Ideal Days, and

later unitless Story Points.[1] The advantage of story points over estimating in absolute time is that it allowed you to calibrate your estimates with what actually happens. We'll take another look at this calibration in Calibrating to Unknown Context, on page ?.

Planning Poker

James Grenning invented Planning Poker[2,3] not to emulate Wide-Band Delphi estimation, but to get a stalled meeting moving again. While he continued to use and teach it, it was when Mike Cohn publicized the technique in *Agile Estimating and Planning [Coh05]* that the practice was widely adopted. Soon many people came to believe that Planning Poker was *THE* Agile estimation technique, though James has gone on to use other methods.

Counting Stories

Back in 2008 or 2009, Bob Payne and I were noticing that teams we were coaching spent an awfully long time estimating their stories in story points. Even though they spent two or more hours every two weeks doing this, the estimates didn't seem to jibe with reality. Figuring that the primary reason for estimating every two weeks was choosing how much would fit into the two weeks, we did some analysis of the data we had and determined that a count of the stories had as much predictive power (or a little more) than did these laborious estimates. We presented a session[4] at the Agile 2012 Conference showing our conclusions. We were not, of course, the only ones to discover that counting stories, especially for the short run, was as good as estimating stories and a lot easier.

Of course, biting off a bite-sized chunk of work is only one aspect of looking ahead. Figuring out when some major functionality is going to be done is another common aspect. Chet Hendrickson tells me that they started with stories to fill eleven 3-week iterations, or about 7-1/2 months. That's a lot of stories to pin down at the start of the project. The C3 project had an onsite customer who knew that much about what needed to be done.[5] I've witnessed a lot who don't, whether or not they thought they did. That's made me very wary of starting with A Large Number of Small Parts, on page ?, as that also seems like a waste of effort for the value returned.

1. https://ronjeffries.com/articles/019-01ff/story-points/Index.htmlurl>
2. https://wingman-sw.com/articles/planning-poker
3. https://wingman-sw.com/slides/Beyond-Planning-Poker-v1r1.key.pdf
4. http://idiacomputing.com/pub/Agile2012-What%27s%20the%20Point%20Of%20Story%20Points.pdf
5. http://www.coldewey.com/publikationen/conferences/oopsla2001/agileWorkshop/hendrickson.html

Definitions

In order to communicate more effectively, it would help to build a common understanding of some terms. I see many arguments that seem to revolve around what is, and what is not, an estimate. In the noun forms, a dictionary gives us the following definitions:

Estimate
: an approximate judgment or calculation, as of the value, amount, time, size, or weight of something.

Forecast
: a prediction, especially as to the weather; a conjecture as to something in the future.

Prediction
: something declared or told in advance.

Conjecture
: an opinion or theory without sufficient evidence for proof.

Projection
: calculation of some future thing, usually based on past results.

Guess
: an opinion that one reaches or to which one commits oneself on the basis of probability alone or in the absence of any evidence whatever.

These words are near synonyms, though some people assign specific differences to them in their own use. Estimate, conjecture and guess are all very general, applying to the past, present, or future, whereas forecast, prediction, and projection specifically apply to the future. I might estimate how many jellybeans are in a jar, but I might forecast how many will be left tomorrow after a children's party. Some domains prefer a particular word; both weather and company profits are generally foretold as forecasts. Future state is often termed a projection when generated mathematically, and prediction when it is not.

I'm fond of saying that the abbreviation of "estimate" is "guess." That often gets a chuckle, because the general feeling is that people are pretty bad at estimating, and particularly bad at estimating software projects. It seems that grabbing a number out of thin air has as much chance of being helpful as anything.

Beyond that joke, estimates are distinct from guesses. At worst they are "educated guesses," based on slim knowledge or experience thought to be relevant. Sometimes the application of past experience is very seat of the pants, with insufficient consideration of the details. With a little more attention to what's similar and what's different between now and past experience, estimates can become truly useful.

In conversation, it appears that many people have a more restricted concept of an estimate. They often associate it with whatever type of estimate has been most prevalent in their experience. For software developers, the most prevalent form is usually "when will you be done?" Given that this when question is often asked before knowing *what* will be done, it's understandable that the word "estimate" has a bad connotation for many software developers.

Nothing about estimate or forecast states how either is formed, but many people closely associate estimates and forecasts with the technique they most frequently use, or see used. A weather forecast may be based on the amount of pain felt in the knee, knowledge of last week's weather, or on a complex model of atmospheric humidity and pressure. A software delivery estimate may be based on knowledge of a similar project, on a detailed decomposition into tasks that are independently estimated, or on a model of project attributes.

Who This Book Is For

You may find this book useful no matter what your role in software development. In fact, you might apply the ideas outside of software development, though I haven't specifically addressed that. When writing this book, I've had three broad classes of people involved with software development in mind. They are as follows:

Software Team Members

These are the people who often get asked for estimates they don't care about. These are the people who often get blamed when results aren't the same as the estimates.

Upper Management

These are the people who have business decisions to make and who need forward-looking data to make them. These are the people who aren't close enough to the development work that's going on to directly know how well things are going.

Middle Management

These are the people caught in the middle of the other two groups and catch it from both sides. These are the people with a responsibility to meet the expectations of upper management and keep those expectations reasonable. These are the people with a responsibility to intervene when needed to help the development effort go well.

All of these people might gain an understanding of some new ways to estimate. They can learn to avoid some of the common pitfalls of relying on estimates with insufficient awareness of their limitations. They can gain some perspective of what needs require what sort of estimates.

All of them might benefit from stepping back to a larger view of the topic, and especially by considering the view from a variety of points of view.

Goal of This Book

The goal of this book is not to make you an expert at estimation. Estimation is a tool rather than a goal in itself. Maintain focus on the real goals, the outputs, the outcomes, and the impact of the system you're building. Estimation is a tool for achieving these goals more reliably in an uncertain error-prone world. Estimation lets you see into the future and make decisions about the path you're on before you reach the endpoint. We'll also take note of some other ways that estimation can help us, as an organization working together, achieve common goals and fulfill the responsibilities of our various roles.

Steve McConnell in *Software Estimation: Demystifying the Black Art [McC06]* and Capers Jones in *Estimating Software Costs [Jon07]* approach estimating toward getting an answer most close to the eventual actuals. Looking at it another way, they try to estimate a duration and cost that project managers can meet. To do that, they recommend such things as collecting data from past projects and making sure the work being estimated is clearly defined. These preliminaries may be beyond your ability in your current context.

This work is focused on helping you make the most of what you have available. It's about being able to make prudent choices for a successful project rather than winning a prize for coming closest to guessing the number of jelly beans in the jar. It defines a successful project by the desirable outcomes achieved rather than by conformance to prior plans. The view of this book is biased toward iterative projects that are started with the knowledge that information is incomplete and are steered toward desired outcomes as they are developed. Estimates are a valuable tool for learning more as you proceed.

To be honest, plan-driven projects, which "plan the work and then work the plan," tend to start with known incomplete information and be steered toward desired outcomes as they are developed. The difference is that on plan-driven projects the iterative work is generally hidden from official view. During implementation, design corrections are made. During testing, implementation corrections are made. Throughout, new or clarified requirements affect all stages of work. The plan-driven bookkeeping hides these iterative cycles performed in the "wrong phase" of the life cycle, or it calls them errors. I call them reality.

Approximations

Often people are expecting estimates to give them precise and accurate values that match the future, as if it's a process of calculation rather than estimation. This leads them to disappointment in the results, and often induces them to make bad decisions along the way. There is a simpler and healthier path you can take.

Before I got into software development, I did electronic repair, and then electronic design. The book that taught me how junction transistors worked was *Transistor Circuit Approximations [Mal68]*. What made this book work so well for me is that it started with the concepts of an "ideal transistor" (or "ideal diode") for a first approximation of circuit analysis. Sometimes second-order effects are included for a "second approximation." These approximations are good enough to understand how a circuit works. In reality, there is much variation from one transistor to another, so transistor circuits are designed to make the circuit performance almost independent of the transistor characteristics. In Malvino's words, "exact formulas for transistor circuit analysis are of limited value to most of us because the exact characteristics of a transistor are seldom known."

This is an excellent way to think about estimation, too. Ignore the second-order effects until you need them. Most of the time, the first approximation will do most of what you need. This book has no magic to give you precise and accurate predictions of the future under unknown conditions. Instead, it gives you the means to get "close enough" to understand what's going on. And it gives you tools to calibrate those tools for your own specific context. Along the way, I suggest some ways of working to make your outcomes less sensitive to the vagaries of inaccurate estimates–ways of achieving success despite the uncertainty of the path to get there.

What's in This Book

Ordering the content of this book was difficult for me, and I rearranged it numerous times. Writing it for a broad audience who are facing radically different situations prevented me from imagining a simple story that starts at the beginning and travels an obvious path to the ending. In the end, I think I have found a story that most will be able to follow, though there are many cross-references to entice you to read things out of order.

We start with *why*, and then proceed with *how*. We repeat this pattern twice, once for anticipating work and once for the the situations in the middle of the work. This is followed by how to handle the disappointment of an inaccurate estimate, and how to make that not a disappointment, but an opportunity. We end with a look at the people issues surrounding estimation. If you're looking for something in particular, the following chapter descriptions might help you jump right to it.

Starting Something New

We start before the beginning, when we're considering starting something. What questions do we need to consider before we start it? There are many different circumstances, and we explore a number of different needs. Likely your need is covered in this chapter. Or some of your needs, though you may have others I've not described.

Comparison-Based Estimation

The basic approach to estimation is comparing something unknown to something known. This is your most likely option when you're starting something new. There are lots of ins and out and wrinkles to this topic, though. This chapter looks at a lot of them.

Decomposition for Estimation

Often it's easier to estimate things if we break it down into smaller pieces. What way do you break it down? This chapter explores a number of different ways by several criteria. It offers opinions for the general case, but you must decide what fits your specific case.

Checking Progress

Once we've started, we start wondering how we're doing. That might be a concern about the finishing date. Or it might be a more general concern if we're on the right track. Sometimes there are concerns about efficiency or

speed. This chapter explores ways of checking progress and when it make sense.

Model-Based Estimation

Direct comparison is the root of estimation, but it we're going to be doing it repeatedly, it's easier if we create a mathematical model of our progress. This lets us recalculate whenever we want, which is important when we're checking our progress. This chapter describes a number of different ways to construct such a model.

Estimating Milestones

The end of the project is not the only date that matters. You'll have interim targets you want to meet, also. There are reasons that these dates might be important for others. This chapter takes a look at these milestones and reasons.

When Estimates and Actuals Differ

Inevitably, what actually happens will differ from your estimate to some degree—sometimes to a large degree. What should you do when that happens. This chapter will give you advice and guidance.

Planning for Incorrect Predictions

Since we're pretty sure that some of our predictions will be incorrect, let's plan for that. Let's figure out how to make use of those incorrect predictions to help us reach our final success. In fact, as this chapter suggests, you might want to create extra predictions just to take advantage of the ones that are inaccurate.

When People Clash

When most people talk about estimation, they talk more about the conflicts between people than about the estimation process itself. This chapter looks at how these conflicts get out of hand, and what you can do to make things better.

Conventions Used

Some of the concepts and lessons around estimation can be noticed in our everyday lives. Where possible, I'll use mundane stories that I hope you will identify with easily. Some of these stories are fictionalized, but they're all based on personal experience.

Other stories are more specific to the business of software development. Since the needs being met with estimates vary greatly with the context of those needs, I'll be using several fictitious companies to represent some of these contexts. These ersatz companies vary in size, focus, and organizational structure. They give us some structure for viewing estimation in the light of concrete details. These examples illustrate the more general principles that govern the needs and practices of estimation. While there are many more possibilities than these three organizations, these examples should help you understand how to apply the principles to your situation.

Empire Enterprises

Empire Enterprises is a large diversified company with a centralized Information Technology department. The software they create is predominately for internal use within the company. Some of it handles the all of the corporation's accounting processes. Other systems support the work of the varied business lines which are the focus of the other company divisions. It's a continual juggling act to be responsive to the different divisions while keeping a focus on what provides the most benefit to the company as a whole.

Riffle & Sort

Riffle & Sort is a medium-sized business that does data processing for other companies. They started out in the 1980s providing payroll services for small businesses. As the business grew successful, they found that calculating payroll for their clients was getting to be more work than they could handle. Fortunately the president's son-in-law had become interested in personal computers, and took on the task to automate some parts of the work. That grew over time, and now the company offers custom software solutions to aid other clerical paperwork needs.

TinyToyCo

TinyToyCo has translated their online game, in which the player tries to keep a virtual cat happy with minimal cat treats and scratched hands, into a phone app where the cat clamors for attention at random times while you're doing something else. Now they want to create a physical version: a robotic lap-seeking Fluphy Kitty™ toy.

Now It's Your Turn

Each chapter ends with a challenge to put what you've learned in that chapter into practice. Try these questions as a thought experiment. If they don't quite

fit your situation (maybe you're not currently on a project), then modify them to work for you.

There is no answer key in the back of the book. These exercises are for you to stretch yourself. Share them with your colleagues and compare your answers. Return to them some time in the future and see if you might answer them differently. Return to them to try them out in a different context. Ultimately, the questions are more durable than the answers.