ESTIMATES

How to measure project progress without estimating

Vasco Duarte

What others are saying about NoEstimates, The Book

"The way this book was written helped me not only to understand the main ideas behind #NoEstimates but to internalize the concept with the real examples behind the story." – *Hermes Ojeda Ruiz, LogicalBricks*

"Finally! The most meaningful text book about project management and governance! This piece of project fiction will become the real "don't panic" guide to project management!" – *Tapio Järvenpää, Beer Executive Officer, Hops United OÜ*

"Anyone interested in deliver their projects on time should read this book. It´ll make you forget the idea that you can predict the future with a crystal ball. *Instead of that you´ll learn to use your data and be more effective. Take of red pill!" – Emiliano Sutil, Project Manager al Xeridia S.L*

"You're doing it wrong. Estimates that is, and #NoEstimates tells you why. Most importantly, however, the book tells you what you should be doing instead. The compelling mix of theory with story means you'll quickly learn the lessons and start benefiting from the book. Read, think and apply #NoEstimates today." – *Dr. Bruce Shcarlau, Senior Teaching Fellow at University of Aberdeen*

"I was sold on this book after reading the table of contents of the first chapter." – Nick Mein, Engineering Manager, Trimble Navigation.

"Do the most important thing is not a startup thing only! In my 15 years of experience with software development I've seen a lot of patterns of projects failing. I've experienced myself every example described in #NoEstimates. I've worked for a startup, where it is obvious what's important without estimating. With the help of the #NoEstimates book I can do the same within a multinational corporate organization*." – Jeroen Knoops, Software Developer*

"Trying to be more predictable by working with estimates is a seductively simple idea. After decades of widespread use of this idea, actual results are now very clear. Estimation has instead caused reduced predictability, lowered quality and unhappy customers. Read this book to start exploring some more successful strategies!" – *Henrik Berglund, Agile Organizational Coach, Cedur AB*

"A must read for all of us that has discovered that it's in the doing of the work that we discover the work that we must do." – Anders Lundsgård, Senior Engineer and Solution Architect, Sweden

Purchase the book at
http://oikosofySeries.com/noestimates-book-order

Buy Now!

NO ESTIMATES

How To Measure Project Progress Without Estimating

Vasco Duarte

NO ESTIMATES

Publisher: Oikosofy Series, www.oikosofyseries.com

Illustrations by Ángel Medinilla
Cover design by Sebastian Hermida

CONTENTS

FOREWORD

"Transformation comes more from pursuing profound questions than seeking practical answers."

– Peter Block

I'm pretty good at getting comfortable in my ways. Perhaps we all are. Still, I'm certain this is not a good thing so I work hard to keep an open mind about making things better. I suspect that our biggest problems are hidden within the things we trust the most.

A few years back I noticed there are a number of practices and techniques used in the management of software development done "by default". That is, we do them without question. Perhaps we assume that this must be the "right way" since everyone does it this way. Maybe it's how we were trained for our job, or that we've been doing it so long we don't even remember why we started. It could be it is a practice someone has documented in a textbook, or we learned through some training sessions at a conference on "how to manage software development." The problem for me isn't whether the practice is good or not, but rather that many often accept it as the only way and don't question its applicability.

Simply following a practice or methodology by default indicates to me that there are likely some hidden problems we might not be willing to uncover. Are we being willfully ignorant, or are we unaware that we are missing something here? When the feeling of control and certainty is more important to us than knowing the truth, and the fear of change holds us back from seeking better, it's time to do some questioning. It's time to face our fears.

"Humans are allergic to change. They love to say, 'We've always done it this way.' I try to fight that."

– Adm. Grace Hopper

Although I love that quote from Admiral Hopper, I'm not so sure that humans are allergic to change. I actually think we are pretty good at it, and in some things we welcome change once we get the

idea that change is needed.

Estimates are one of those things where "we've always done it this way", and there is little scrutiny of the use of estimates. When estimates don't seem to be serving us as we wish they would, the natural or common approach is to simply try to get better at estimates. As business people (and as humans), we need to have confidence in the methods we use - "getting better" is a less than meaningful approach unless we have sufficient faith in the thing we need to get better at. And even still, we must keep a close watch on whether things are getting better overall, and not just the estimates themselves.

So naturally, I want to dig a bit deeper. Estimates themselves are not bad, and in some situations they can be very helpful. However, estimates are often part of an unquestioned process of managing software development, so I want to learn more.

When I become aware of a situation like we have with estimation I want to question, explore, and validate (or invalidate) the practice or method. Is it suitable for the purpose to which it is put? To accomplish this, I start by asking questions of myself. I want to find the "profound questions" that will lead to a better understanding.

I have a number of questions I've posed to myself about estimates. Here are a few examples of the sort I find useful to ponder as a starting point:

- "If I found estimates are not informing the decisions they are meant to inform, what would I change about the way I use them?"
- "If I found estimates are not useful for a purpose, in what way could that be the fault of the estimates themselves? In what way is that an issue with the purpose?"
- "In what way would getting better at estimates help if the estimates themselves are of questionable use?"
- "What ways can we meaningfully validate decisions we've made based on estimates?"
- "If we believe that getting better at estimates is our only choice,

what are the possible pitfalls?"
- "If the decisions that we currently think are important are not as important as we think they are, what should we do?"

Early October, 2012: I had just met Neil Killick in Twitter, and read a blog post of his named "Should We Estimate Software Projects… At All?" After a few Tweets back and forth, Neil introduced me to Vasco Duarte. I believe that the first blog post from Vasco that I read on this topic was "A better way to predict project release date!" What a great starting place. Alone, I had my suspicions about estimates, but with others such as Neil and Vasco also questioning the practices of estimation and use of estimates I was encouraged that some real progress can be made.

So now there were three: Neil, Vasco, and myself. Little did I know how much I would learn from my on-going conversations with these two. Even more wonderful was that new folks were joining the conversation daily. One notable luminary is Ron Jeffries who has now written extensively on the topic, and over time many others such as Allen Holub and Kent Beck have shared their ideas on the topic. But I'm jumping ahead.

I noticed an interesting thing very early on: While we were all questioning the "de facto" use of estimates in managing software development, we had different ideas about what the core problems might be, and what to do about them. I took this as a good sign. I seek a broad spectrum of ideas, especially at the beginning. We were all noticing that something was amiss.

If the use of estimates and estimation are suspect, I'm more interested in learning all about that than what to do about it. The "how" reveals itself when we understand the "why" about the "what." This is where the power of pursuing profound questions comes into play for me. The "how" of estimates is a moot point if they are not serving us well. It's of little value to solve a problem we don't yet understand; we are just a likely to be addressing a symptom rather than the underlying problem. Let's get that settled first, and then explore the alternatives.

Getting better at estimates (which the entire industry has been trying to do for a long time) doesn't help if the underlying motivation for using them is not valid. Rather than jump to the possible solution of getting better at estimates, I want to do whatever it takes to understand the underlying reasoning and purpose of the estimates. That is an area worth exploring.

Along the way it became apparent to me that some of the decisions we depend on for managing software development efforts are probably not the right decisions to be making. Why do we feel that knowing the cost of something "up front" is useful? Is that really important, or simply just something we've convinced ourselves that we can't do without? Why is it that we think we have "limited funds" for software development? If we are good at generating income from the work we do, are these limits even meaningful at all?

What if there are alternative models that lessen or eliminate the need for knowing the cost? Consider that it might be better to become great at creating software, and delivering it into actual, meaningful use early and often, and earning a return almost immediately. Would knowing the cost be as important if we can start re-cooping whatever costs there are almost immediately? Maybe by following this model the desire and interest in estimates will simply fades away. Maybe not. Let's find out.

Estimates are used in many ways, and there are seemingly countless methods of estimation for devising those estimates. What if we could forecast some aspects of our project without estimates at all? This is something I find interesting, and it's a possible game-changer.

Vasco has been exploring one such alternative that is of particular interest to me, and he covers it nicely in this book. His ideas about forecasting based on the data collected in the doing of a project eliminate the need for certain estimates for that project, and the results are easily verified. If we find that forecasts are helpful, what if we could find a way to forecast without estimates that works at least as well as estimating does? What if it is actually better? And easier? How would that change our dependence on estimates? How would that change our regard of estimates? Should we at least

experiment with ideas like this? What if we could try this and prove it for ourselves while still working in the same way we always have? Powerful stuff.

I'm glad you are reading this book, and exploring along with Vasco. While this is still a very new effort, many people are gathering around these ideas and questions, and exploring for themselves the method that Vasco shares in the #NoEstimatesBook. Business people, Managers, "Customers", Developers, Analysts, and just about anyone will find this book useful. We all see a need for better, and insist that regardless how well entrenched the "way things are" might be, it's time for a change. And we welcome it.

Woody Zuill
#NoEstimates Pioneer, and Software Industry veteran
www.MobProgramming.org,
http://zuill.us/WoodyZuill/

THE #NOESTIMATES PREMISE

If you have been around Agile and Lean for some time you might be familiar with the Japanese-born concept of "no inventories". Between the sixties and the eighties, Japanese companies – Toyota being one of the most prominent examples – embraced a new paradigm of production founded on the premise of maximizing value for the customer and seeking continuous improvement. In order to maximize value, all activities and investment that were not directly adding value to de customer were labeled as "waste". In this sense, moving materials from one place to another is waste; reworking, overworking or overproduction are waste; fixing defects is, of course, waste; and unfinished materials waiting in a queue to be used (inventories) are waste.

The golden rule to determine if something in your organization falls in the category of waste is: "do we want to have more of this? Like double or triple it?"

Companies seldom want to ask this golden question from themselves. If you ask it too much, you'll find that Managers are waste - you don't want more of them. Meetings are waste. Code is waste.

But, if mostly everything around us is labeled as waste, shall we get rid of it? The answer, of course, is no. The premise is that you want to have as little waste as possible, and not more. Once you reach the state of minimum waste, you should try to go even further and reduce it a little bit more – continuous improvement or Kaizen.

Of course, you don't want to double or triple your inventories. Inventories are a hidden bank account, money lying around doing nothing but deteriorate. Even worse, it requires investment in the form of space, maintenance, rotating stocks, security… You even lose inventory pieces due to aging, bad storage or warehouse mismanagement. You have to maintain stock records and reconcile inventory periodically – and if you've ever done something similar, you know how soul crushing this process can be.

So what's the only reason we keep inventories? Organizational dysfunction, or so believed the Japanese companies. People need inventories so they have access to materials between resupplies. These resupplies were separated by several hours, sometimes days. Each resupply needed to be huge so workers would have materials until the next resupply. Huge resupply needed space, storage, management, rotation, movement… Waste everywhere.

The Japanese breakthrough was to imagine a company where resupplies were happening constantly, Just In Time. By imagining how work would look like in a world with no waste, they could forge new production paradigms like one-piece-flow, single-minute-exchange-of-die, zero-defects or total-quality.

Of course, you could be picky and observe that each worker on a Lean factory might have a small inventory of pieces around, maybe a handful, maybe a dozen. You could then claim "Look! Look! That's inventory! It is not actually true that they work with #NoInventories!" The point is that these Lean factories have been able to reduce the resupply period from days to minutes, and their stock, inventory or storage needs from stadium-sized warehouses to drawers.

So, #NoEstimates…

It is obvious to me that estimates do not provide more value to the customer than inventories. People who find value on estimates are just addicted to a practice (estimation) to get something they find valuable: plans, comfort, uncertainty reduction, financial projections, sales proposals… But, again, these are not customer value. Applying the golden question, you don't want to triple plans, estimates, reports and proposals. Hence, you want as few as possible – not less, but not more. So they immediately fall under the label of waste.

Of course, you will always find someone that defends that inventories are in fact customer value, the argument being that inventories protect the customer against accidents and company dysfunctions. No serious Lean discussion can be forged around this

argument nowadays.

Interestingly enough, when it comes to reducing the need and dependency on estimations, people can be quite belligerent – you just have to wander around the twitter conversations around #NoEstimates to get a taste of it.

I imagine that, in the early days of Lean, when Womack and Jones were just starting to spread the news about this crazy Japanese production system, several people would be as baffled as some other are nowadays with the #NoEstimates question. I can clearly imagine western managers protesting, "just in time production? Resupplies every few minutes? Are you CRAZY? Why are you trying to debunk the proven practice of accumulating inventories? What are you, smarter than all of us, than all managers that have been promoting stocks and inventories for the last four-plus millennia?"

Ah, human beings… 'gotta love 'em.

I have faith and patience. When some people in the Agile community started talking about zero defects or continuous delivery, not few voices rose against such concepts stigmatizing them as 'delusional' and 'misleading'. Even the ideas behind the manifesto, which are enthusiastically defended nowadays, were difficult to promote mostly everywhere just ten years ago.

I feel there's a natural learning curve towards #NoEstimates. It is very frequent nowadays to hear from teams that moved from WBS-like estimation - estimating each and every task, then adding up all estimates - to story estimates, point-based estimates, t-shirt sizes and, counting stories and, finally, dropped the estimation-based conversations to focus on throughput, cadence and flow.

I did my first public talk on #NoEstimates at Agile Lean Europe Berlin 2011. The audience was crowded by several well-known European Agilists, and I could metaphorically hear the sound of their neurons crashing into each other while I presented the idea of reducing the estimation effort and focus on stabilizing the team throughput and using past information to drive forecasts. People

rapidly organized an Open Space around the topic, and the usual question was "yeah, but… You need estimates, right?" Not a question, in fact, more of a desperate cry in the middle of a withdrawal syndrome. My answer was direct – no, you don't need estimates. You are using estimates to produce some other stuff, so the question is, can we produce that other stuff in a better way that using estimates?

Unfortunately, several people still ask for better ways to estimate. It is a flawed question. For me, it is like asking for better ways to win the lottery, or better ways to guess what team is going to win a sports match. But I have faith in the capacity of human beings to change, and patience to watch that change happen.

A year ago I decided I wanted to push this change a little further and I asked my good friend Vasco Duarte if he was up to the job of writing a first book on #NoEstimates. Vasco had enthusiastically defended this approach based on his own experience. A few months after my talk at ALE2011, he presented a similar topic at OOP2012, and have been very active on the #NoEstimates debate since the very start.

We started this work together, but I had to quit after some months – my job engagements and workload were keeping me behind the intended pace, and I was basically endangering the project. To be honest, we also had "creative differences", as artists say, and decided to "pursue our own careers and personal projects". No problem. We are still good friends, and I kept illustrating Vasco's work - sort of – and providing my feedback and insights. We still collaborate on some other projects, and I'm sure we'll keep finding opportunities to promote innovation on the Agile world.

My final word to you, reader of this book, would be to remember the fact that mind is like a parachute, it only works when it is properly open. Too many people have lost the track of the #NoEstimates premise to dig into inane details. For instance, some people would say "oh, but you are forecasting results based on past information – isn't that an estimate?" Or maybe "oh, but you are breaking big stories into small, one-or-two day sized stories, but in

order to do that, won't you need to estimate the story size?" Remember that #NoEstimates is not about no estimation ever, but about the minimum amount of estimates that will do, and then look carefully at ways to reduce that need even more. In that sense, I love J. B. Rainserger concept of #MehEstimates, like in "Oh, estimates… Meh!... Yeah, I guess we're doing some, we are just not very concerned nor serious about them."

So, here's to life, the #NoEstimates book. Enjoy.

Ángel Medinilla
Agile Coach, #NoEstimates practitioner, Book illustrator
http://www.proyectalis.com/

CHAPTER 1:
THE PROBLEM WITH ESTIMATES

Carmen came into the office bright and early in the morning. She was surprised when her boss unexpectedly called her into his office. Any other day he would greet her; invite her into the break room for coffee; have a friendly one-on-one chat with her, and talk about life… But not this time.

This time her boss looked nervous, and he closed the door before he even said good morning to her. Carmen appreciates her boss, because she has learned a lot from him. But right now she is nervous. What could her boss possibly want that required them to meet behind closed doors? What could be so secretive that it couldn't be overheard by anyone else in the office?

"Carmen, we have a big project coming in. This is potentially the largest software project our company has ever won."

"A big project you say?" Carmen's interest grew. "It's about time they consider me for a big project," she thought.

"Yes! This project alone could finally make us profitable! And I want you on it; I trust you and appreciate your work greatly. I know you can pull it off…"

Carmen was surprised. She had heard of a new, secret client project that they called Big Fish, but she did not expect to be assigned to this project. After all, she still had the Telemark project going on.

Carmen shifted in her chair and tried to think of something to say. Something didn't feel right.

"Well, that's a surprise. I still have the Telemark project, which will be landing soon. I thought you wanted me to finish that before moving on to other projects…"

"I did, but his came up suddenly." Carmen's boss interrupted, "I only just heard about it yesterday. Our CEO was able to find a way for us to bid on this project. It will be huge!"

"Errr, Okay. What should I know about this project before I take this assignment?" Carmen was unsure about this. Her boss seemed almost too eager for her to say yes to the project.

"Well, basically nothing. It's just another software project. The only difference is that this project is bigger than the others you've worked on. But this time it's with a very more important client."

"How much bigger project are we talking about?" Carmen asked.

"Well…" She felt uncomfortable. Her boss wasn't being upfront with her.

"Is it twice as big as the Telemark project?" She asked, but her boss didn't reply. "Five times bigger?" Still no comment. "Ten times bigger?"

"Actually, it is a public sector project, so I guess you could say that it's several orders of magnitude bigger than your Telemark project…." He finally admitted.

"Oh…" Carmen was speechless. This could potentially be the largest software project in the country, and she was being lined up to manage that project.

"And we'll need to present a bid by the end of next week!" Her boss looked excited. "So get your team on it. We need to close this deal now Carmen; it could change our company's future! We need to submit our proposal as soon as possible!"

Do Estimates Work?

If you were Carmen, you'd probably feel nervous too! I know I would. Estimating a software project is never an easy task, and the situation gets even worse when you have to do it without first understanding the context or the requirements well enough. Add that to the pressure to create a winning bid, and you have an explosive mix! And not in a good way.

When I talk about #NoEstimates I get the most serious questions from people like Carmen. Questions such as, "How can I win a bid if I don't estimate?", or "My customers will not award me the contract unless I give them an estimate!"

Fair questions, but before we tackle them, let's tackle the real question here; a key question for both you and Carmen, which is *Do estimates work?*

Some researchers have already proposed what a *"good"* estimate should be. In 1986[1], they proposed that a good estimation approach would provide estimates *"within 25% of the actual result, 75% of the time"*.

What this means is that, for all the projects you estimate, you should complete those projects within 25% of the original estimate for 75% of the time. Another way to put this is that you don't have to always be within 25% of your estimate, but that you should be for 75% of all of your projects. Let's examine the track record in the IT industry to see if we're consistently able to estimate correctly.

First we look at the Chaos report, a study that is run every few years to assess the *"success"* of software projects worldwide:

---

[1] Steve McConnell, *Software Estimation: Demystifying the Black Art*

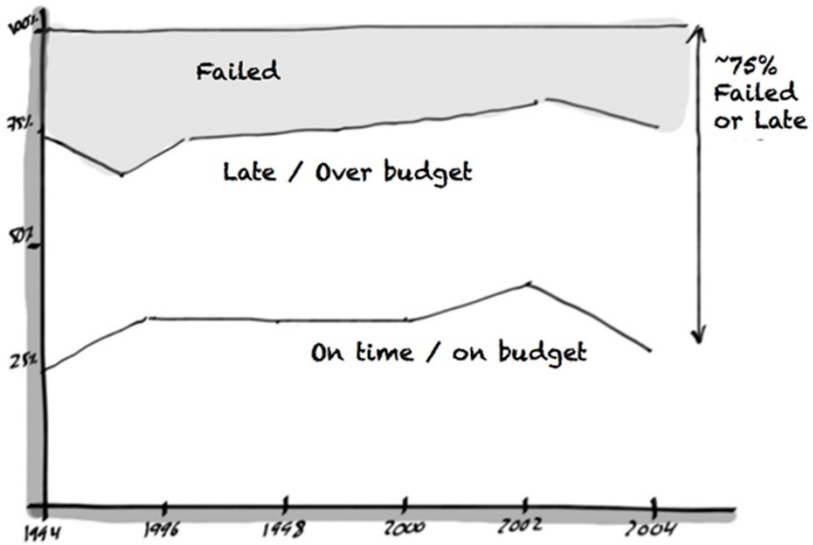


Figure 1 – Project success rate over the many CHAOS report surveys from 1994 to 2004. Graph adapted from Steve McConnell's book, Software Estimation: Demystifying the black art

This picture looks grim. If Carmen works for an *"average"* company her chances of success are slim, about 25% if you believe in the accuracy of the Chaos report. Not good.

You can improve your chance of success by using certain software development practices like Agile, but even then the success rate is a poor 42%[2].

But let's look at this from another angle. Even if you agree that being within 25% of the original estimate is a good result, what about the outliers? The tasks and projects that exceed the original estimates by so much that they run the chance of jeopardizing the entire project or portfolio? You may think that your projects don't have such tasks.

…Bad news…

---

[2] CHAOS report for 2011-2012 shows 42% successful agile projects.

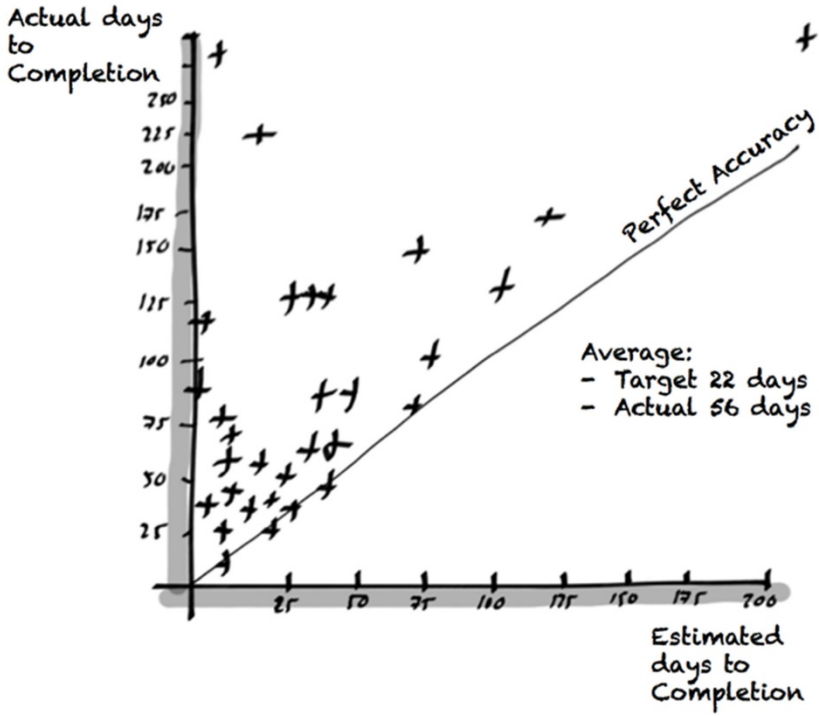


Figure 2 - Project estimates vs. Actuals (in days), track record for one organization. Graph adapted from Steve McConnell's book: Software Estimation, Demystifying the Black Art.

Look at the project on the top-left of this chart. This project was estimated to last 7 days, but ended up taking more than 250 days to complete. This is a difference of 1 week to nearly 9 months! While this is only one outlier, but the potential impact on the company is tremendous. A project like this can drive a company bankrupt.

There are several examples in our industry that look like this. In a famous UK contract walk out, Accenture avoided up to £1bn in penalties[3] in a contract with the National Health Service (NHS) in the UK. Some will argue that this is an extreme case, an outlier. But do you want to bet your company on one project if the possible

[3] Accenture avoids £1bn penalty for walking out of a contract with the NHS http://www.theregister.co.uk/2006/09/29/accenture_nhs_penalty/

result is obliteration? I don't think so!

Beyond the problem with outliers, let's look at examples of sustained performance. Here's the project delay (in % from original estimate) for 17 different projects from one single company spanning a period of 4 years.

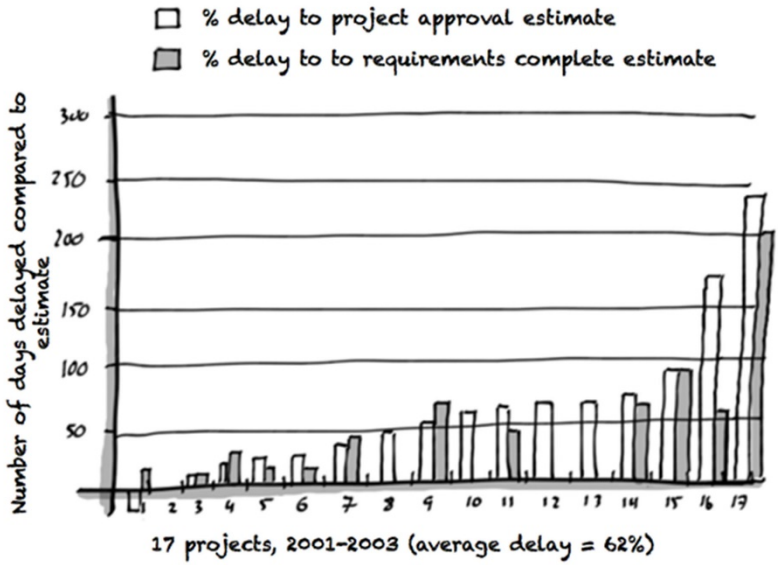


Figure 3 Percentage delay to estimates at approval, and estimates at requirements completed phase for one organization. Adapted from data collected by me in one company between 2001 and 2003 for 17 different projects.

This company had an average (yes, average) delay of 62% for their projects. This means, for every project that should have lasted 3 months, it lasted nearly 5 months! As I look at the graph above I can't help but ask: *"Can I really trust estimates as a reliable tool to manage our companies?"*

Would you bet your entire company on a work-method (estimates) that delivers 60% to 80% of the projects late or over budget? This is the question that Carmen had to ask herself in the scenario above. Can she, in all honesty, risk her company's future on a method that has that large a failure rate?

Why Don't Estimates Work?

"Carmen, I want you to know that you have my full trust. I know that you can pull this project off."

"Thank you sir, that means a lot to me." Carmen said, while thinking: "and I may need to call on that trust. Hopefully you are not bluffing."

Estimates don't really work. Specifically, not in the way the software industry practices estimation. And not in the way that many, including in the Agile community, propose estimates should be done. There are three clear, intuitive, and proven reasons as to why that is so.

Hofstadter's Law

Hofstadter's Law: It always takes longer than you expect, even when you take into account Hofstadter's Law.

— Douglas Hofstadter[4]

Douglas Hofstadter observed that, no matter how much work went into developing computer programs to play chess against Grand Masters, the winning program always seemed to be 10 years away.

Even though Hofstadter was referring to chess programs, this observation was picked up by several other writers, such as Brooks in *The Mythical Man-Month* and Beck in *Extreme Programming Explained*, as a way to highlight just how difficult it is to estimate programming, or other software development related tasks.

However, Hofstadter's Law is only one of several explanations for the continued failure to correctly estimate software development work.

---

[4] Gödel, Escher, Bach: An Eternal Golden Braid. 20th anniversary ed., 1999, p. 152. ISBN 0-465-02656-7.

Parkinson's Law

Work expands so as to fill the time available for its completion.
— Parkinson's Law

Parkinson observed that bureaucracies seemed to expand and therefore create more work for themselves. He was commenting on the British colonial bureaucracy, but the rule was picked up later by Alessandro Natta in Italy and Mikhail Gorbachev in the Soviet Union who famously stated that *"Parkinson's Law works everywhere*[5]*"*.

Indeed, it works everywhere; even in software development environments where the modern formulation of Parkinson's Law is *"work always expands so as to fill all the time available for its completion"*.

Another way to put it is: *"If you wait until the last minute to complete a task, it only takes a minute"*... Or does it?

Parkinson's Law is a description of the phenomenon that work will consistently expand to use all the time allocated to it. In other words, projects that get delayed, rarely catch-up. This, in turn means that if you have one task that is delayed, you should consider all subsequent and dependent tasks to take at least the time they were allocated, meaning that they will end after the originally scheduled time. Therefore, having one task late can potentially derail the whole project. How about that for a shot in the dark?

Accidental Complication

Another reason why estimation is so hard goes by the name of accidental complication. J.B. Rainsberger introduced this concept at Oredev 2013[6].

---

[5] O'Sullivan, John (June 2008). "Margaret Thatcher: A Legacy of Freedom". Imprimis (Hillsdale College) 37 (6): 6.
[6] Watch the full video here (duration ~8 min): http://vimeo.com/79106557.

The gist of the argument is this: every feature or functionality added to an existing project has two cost elements (what you'll want to estimate):

- **Essential complication or g(e):** How hard a problem is on its own. For example, implementing tax handling is hard, because the tax code is complex in itself. That is what J.B. Rainsberger calls essential complication or g(e).
- **Accidental complication or h(a):** The complication that creeps into to the work because – as J.B. Rainsberger puts it – *"we suck at our jobs"*. Or more diplomatically, the complication that comes from our organizational structures (how long does it take to get the approval for a new test environment?) and from how programs are written (no one is perfect, therefore some things need to be changed to accommodate the new functionality).

The cost of a feature is a function of both essential complication and accidental complication:

$$cost\ of\ feature = f(g(e), h(a))$$

In software, estimates of cost are often done by analogy. If features A and B are similar, then if feature A took 3 weeks, feature B will take 3 weeks. But then J.B. Rainsberger asks: *"When was the last time that you worked on a perfect code base, where the cost of the accidental complication (how messy the code base was) was either zero or the same multiple of g(e) as when you implemented feature A?"*

He concludes that often the cost of accidental complication – dealing with organizational and technical issues specific to that code, and that organization – dominates (yes, dominates!) the cost of a feature. So, if h(a) is much larger than g(e) the cost of a feature cannot be determined by relative estimation. In turn, this means that the most common estimation approach, Story Point estimation, cannot work reliably. What this means for you is: You can only determine the real cost of a feature by recognizing and measuring accidental complication, or – like I suggest in this book – by using the #NoEstimates approach.

As Troy Magennis [7]shared with me in an email conversation:

> When we ask someone to estimate how long it will take to develop a feature or project they think about how long it would take hands-on, with perfect hand-off between teams and other staff with the specialist skillsets. This never happens. Work sits in queues; people work on more than one project at a time; people take vacations. We estimate the work, when we need to estimate how long that work takes to go through a complex system given all of its foibles.
>
> Systems thinkers like Goldratt, Reinertsen, and Deming understood this well. Work spends considerable more time idle and blocked by system constraints than progressing towards completion. Given that for inventive knowledge work the hands-on time through a system is considered exceptional at 30%, even if we were PERFECT at estimating that 30% of the time, we would be off by a factor of 70% without compensating. Even the commonly used doubling what you are told method would still be 20% low.
>
> The delays involved in a complex system are unknowable in advance. Just like travelling to work along the same highway each day, you can't know there is going to be an accident blocking all lanes three weeks from now, you can't know that the test environment will "catch fire" three months from now blocking all testing work. It's these unknowable system delays that make forecasting software projects exceptionally difficult.
>
> The only way to estimate and forecasting a system is by estimating the system impacts and the easiest way to do that is by observing prior history of work flowing through that system. #NoEstimates is a strategy that considers the system on its own terms, allowing some ability to predict completion times in advance.

---

[7] Troy Magennis writes about software projects in his blog: http://focusedobjective.com/news/

What Troy refers to is a critical insight: work flows through a system. Without understanding how that system works through metrics like cycle-time, lead-time, etc., the best we can do is estimate a "perfect" engineering day. You and I both know how many of those "perfect" engineering days we've enjoyed in our lives. Not many.

More Than Meets the Eye: Why Estimates Are Actively Harmful for Your Project and Your Company

"Carmen, one more thing." Her boss said as she was about to leave the office.

"Yes?"

"We have a lot riding on this project. It is important that we give them the right estimate for this project!" Carmen did not fully understand the meaning of this statement.

"What does right estimate mean? Why would it be important to have the right estimate instead of whatever else he thinks I may give? Does he want to setup me up?" Carmen's thoughts quickly spiraled in her mind, but she decided to keep them quiet for now, and answered: "of course!"

I would not like to be in Carmen's shoes. What should she do? The average person would first find out what *the right estimate* for the project was, and then deliver that. And at first they would be hailed as heroes. But, what if ***the right estimate*** at the time of the project bid is actually ***the wrong estimate*** in the end?

Deming[8] was quoted as saying that "if you give a manager a target he will do anything to meet that target, even if he has to destroy the company in the process". This is why targets are so important, but at the same time so dangerous. Carmen will probably go to her team, do the best job she can, knowing full well that the goal is not to give the customer a realistic expectation of how much time and money is needed to complete the project, but rather that she is responsible for winning a contract – no matter what it takes.

Wanting to meet the target is a very strong pattern in organizations. So much so that, at the end of every quarter we see a frenzy of meetings and busy people trying to meet that quarterly target, be it sales or operations. Estimates are no different and help make projects prisoners of the targets they set. And this is one reason why estimates can transform from useless but harmless, to influential and harmful targets. Look at what happened at Enron in 2001[9] where the "compensation and performance management system [that] was designed to retain and reward its most valuable employees, [...] contributed to a dysfunctional corporate culture that became obsessed with short-term earnings to maximize bonuses."

But there are more reasons and tempting shortcuts that make estimates actively harmful in certain situations. Below are 4 situations that are possible because of the use of estimates in software development:

---

[8] W. Edwards Deming, American writer, author and coach that helped propel Japan to the most productive nation status with his work after the Second World War.

[9] For details on the Enron scandal, visit http://en.wikipedia.org/wiki/Enron_scandal

- **Internal politics** become a problem when deadlines are set because top management wants something done by a certain date, with fixed scope and fixed resources. Despite advises to the contrary, in these cases the HIPPO (Highest Paid Person's Opinion) overrules any potentially "good" estimate created by the team.
- **Estimate bargaining** occurs when a team has created the estimate and defined a possible release date, and then they suffer pressure from stakeholders to change their estimates. Symptoms of this dysfunction include calls for "more efficiency", or "more commitment", generally followed by phrases such as "we are all in this together".
- **Blame shifting** happens when people outside the project team estimate the work (e.g. experts), and then the project team members are blamed for not meeting those estimates.
- **Late changes** are a form of Estimate bargaining, whereby changes are added to the project, but the schedule is not allowed to change for whatever reason.

These are only some of the possible dysfunctions associated with estimates. This list is an illustration of how estimates can be used and abused for purposes other than actually understanding the project and the work that needs to be completed.

Entangled: How Complexity Destroys Predictability

The biggest conundrum of all however, is how to deal with complexity. How complex dependencies in software projects hide the really big risks.

Software projects present challenges that are unique to the software development domain. Unfortunately, many in the software industry still fail to recognize this. I often hear that "a good software project must, like a house, start on strong foundations of good architecture and good requirements". My usual answer is: "sure, if you consider that you can start by building a tent, then evolve it into a hut, then increment it into a trailer, and later on deliver a castle, then yes, you could say that building software is the same as building a house."

Clearly, software is far more flexible and susceptible to evolution than a house. And that leads to creating a Complex[10] environment where simple *A causes B* causality links are not always possible to determine. Software does not follow simple causality links. Rather, it follows complex causality links. A complex link is, for example, when a small change (like a single line) can cause a catastrophic failure. Take Apple's famous "goto fail" debacle[11], where one simple line of code completely destroyed all the hard work to keep their product safe.

No matter how strong the foundations of Apple's products are, this single line addition to the code was enough to completely destroy their foundations and security credentials. How about that for a house project?

Apple's example above illustrates one additional property of software systems: non-linear causality, i.e. when a small change can have very large impact.

When it comes to estimates, and estimation in general you have to be aware that no matter how thorough you are at estimating the work you must complete, it is unlikely you will uncover all the small problems with disproportionally large impacts. And there's a very good reason for that: we wrongly perceive these examples of extreme non-linear causality to be rare[12]. We perceived to be so rare

---

[10] For the remainder of the book we will use the term "complex" to define an environment where the relationship between cause and effect can only be perceived in retrospect, but not in advance. This definition comes from the Cynefin model (https://en.wikipedia.org/wiki/Cynefin) developed and popularized by Dave Snowden.

[11] For more details on the Apple Goto Fail details, visit: http://www.digitopoly.org/2014/02/26/apples-goto-fail-is-pretty-interesting-and-worth-reading-about/ Read also Uncle Bob's take on the "sensitivity problem" that software development suffers from: http://butunclebob.com/ArticleS.UncleBob.TheSensitivityProblem

[12] In Black Swan, Taleb describes how often we ignore extreme conditions because we don't consider the nature of systems we analyze. He describes 2 types of systems: "mediocristan" where linear causality rules; and "extremistan" where

we usually call them *outliers*. The problem is: in software, an outlier may completely destroy all the effort that went into creating credible, solid estimates.

Does this mean that we cannot estimate? No. But it does mean that our estimates are built on quicksand, no matter how strong we think the foundations are.

Don't believe me? Check out this collection of quotes that emphasize how unstable the ground underneath estimates is.

Berard's Law[13]:
Walking on water and developing software against a written specification is easy if both are frozen.

Humphrey's Law:
The user of the software won't know what she wants until she sees the software.

Ziv's Law:
Requirements will never be completely understood.

Wegner's Lemma:
An interactive system can never be fully specified nor can it ever be fully tested

Langdon's Lemma:
Software evolves more rapidly as it approaches chaotic regions.

Medinilla's Law for Software Contracts:
The 80 000 lines of code won't fit into a 20-page contract.
Do you have some more quotes to share? Send them over, I would

---

non-linear causality is prevalent. He then explains how we tend to classify the systems we study in "mediocristan" because we don't consider the potential for non-linear effects which should drive us to classify those systems as "extremistan" systems.

[13] From: http://en.wikiquote.org/wiki/Edward_V._Berard

like to collect them. Email me at vasco.duarte@oikosofy.com

Quotes about late changes in projects
Thank you Andres Lundsgård for the following quotes.
"A late change in requirements is a competitive advantage." Mary Poppendieck
"The only thing certain in life is change." Anonymous

Change Requests, Change Everything

And then there were changes. You have been there before. You work hard to try and plan a project to the best of your knowledge. The team starts to get the hang of the project and is able to deliver some software, only to see the changes start flowing in. First they flow in at a trickle, one by one. But later on, as the testing progresses, changes come in like an avalanche.

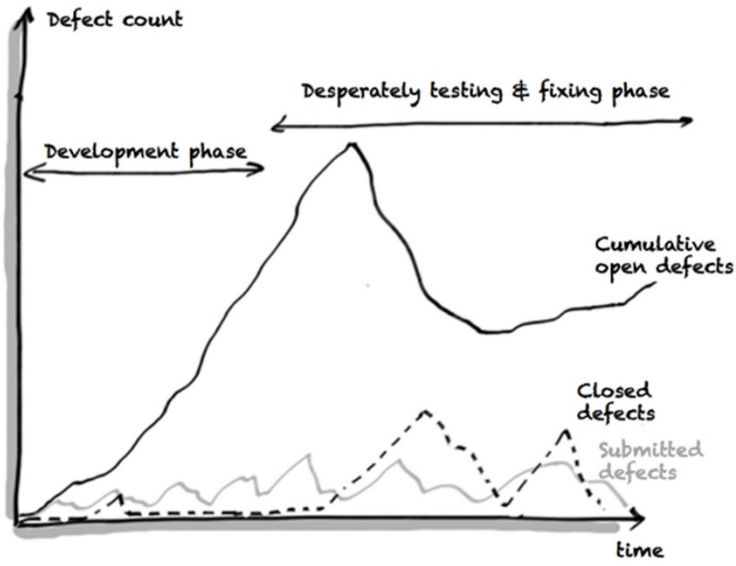


Figure 4 Number of cumulative open defects and the daily rate for closed and submitted defects during a project. Includes indication of development and testing phases. This is an illustration based on data collected during the lifecycle of a project where I worked as project manager.

Hopefully you have some margin (aka buffer) in your project. Right? Sure, but how much of a margin is margin enough? According to the definition of a good estimate cited earlier in this chapter, you should count on at least a 25% margin - assuming the estimation was "good" to begin with.

Are you safe if you add a 25% margin to your project? The answer is no. And if you believe it helps, just re-read the Hofstadter's Law again.

In software development, changes have (sometimes very large) side effects. As you discover problems you find that the consequences of those problems go beyond what some call "bugs", you discover blind spots: complete sets of functionality that require rewriting; or new functionality that was missing. As a friend once described to me: "building software is like when you try to unclog the pipes in your bathroom and end up having to build three more houses just

to get the toilet working again".

Can you predict all the significant changes you will get, and the impact of those changes? If you answer yes please send me your CV. But you will probably answer "no", just like many people do when asked the same question. The bottom line is that estimation is a very poor method to predict the future, and change requests only amplify that problem.

All or Nothing: Death March Projects

Estimations can become a life or death game for teams and companies alike. But nowhere is the effect of bad estimation more visible than in the statistics of mental health problems in overworked and overstressed IT industry folk[14].

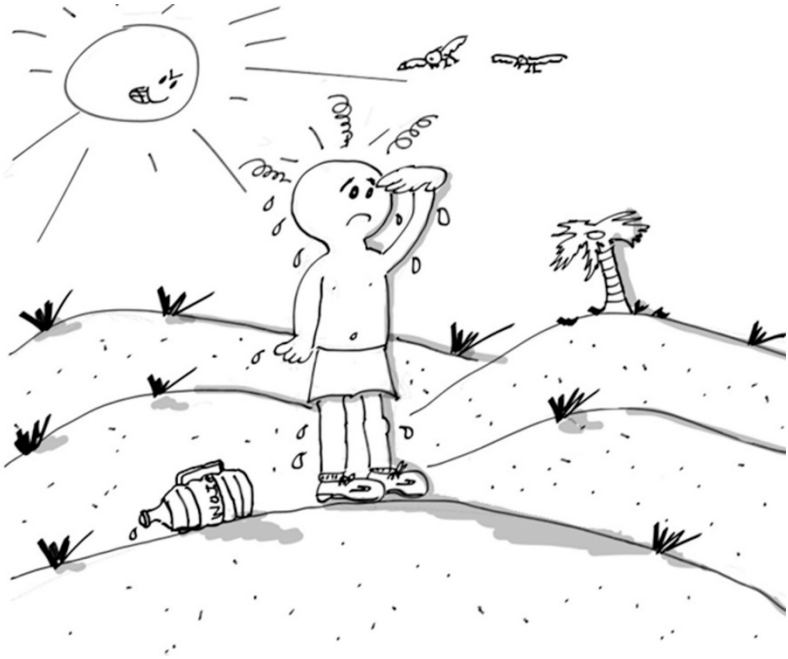
The idea of the death-march project[15] can be summarized like this: you estimate that you need five days to cross a desert, so you pick five water containers, one for each marching day, and you start walking. But, at the end of day five, you are out of water and you are still in the middle of the desert.

What can you do? You cannot go back, as you know that it will mean five days walking through the desert and you don't have any water for that. All you can do is keep walking until either you find more water; you reach your destination; or you die.

---

[14] Article on the incidence of burn out and stress related illnesses in IT http://www.net-security.org/secworld.php?id=14666

[15] The idea of death-march projects was made popular by Edward Yourdon in his popular book Death March: The Complete Software Developer's Guide to Surviving 'Mission Impossible' Projects
http://www.amazon.com/dp/0130146595/



In a similar way, some projects follow an "all or nothing" strategy. This strategy borrowed from poker means that a team either delivers everything ("all") or the project has no value ("nothing"). In this context, the estimation process becomes a battleground between the teams and the customers for that project. The teams want to survive at the end, and the customers want the certainty they will get everything they want – and ***when*** they want it.

At the end of the estimation process the project will start, and the people trapped in that project can easily fall into a death-march mentality. They battle everyday to get the project on track, but a combination of estimation problems with constant change requests transform that project into a constant struggle.

You work every day to the best of your ability and deliver concrete work. But instead of feeling confident and enjoying the progress, you feel like Sisyphus[16]: pushing a boulder up a hill that will

[16] The myth of Sisyphus: http://en.wikipedia.org/wiki/Sisyphus

invariably slide down, again and again. Making you - and the team - feel that there is no hope. A death-march project is caused by the all or nothing mentality and a utopian faith in estimation.

Is There Hope for Software Projects?

If it is so hard to estimate software work, can you predict when a particular project will end? Yes, you can. And that is the right question. Instead of asking how long the project will take, ask instead: *"given the rate of progress so far, and the amount of work still left, when will the project end?"* Or, a similar question: *"Given the rate of progress, how much of the work can be finalized by date X?"*

These are much more powerful questions. Here's why:

- These questions don't assume an all or nothing mentality because, for example, you can remove work from what is left.
- These questions don't assume that it is possible to predict the future. Instead to answer them, you can use *data* about past progress and constantly update your answer to the questions.
- To answer these questions, you can use available *data* that the project team has already collected (aka actuals) in the form of "rate of progress", instead of having to guess at future events without considering the current rate of progress.

In this book we will cover how Carmen will answer the question "when will X be done?" using something else than estimates in Story Points or hours. We will also cover the main challenges that come from the traditional approach to project estimation, and how to avoid, or reduce the impact of those challenges. But before exploring how a #NoEstimates practitioner would predict an end date for a project or how much work can be done by any given date, let's review some of what the software industry today considers "best practice" methods for software estimation. This review will be essential for us to clarify the stark contrast between Estimates and #NoEstimates.

CHAPTER 2:
WHAT ARE ESTIMATES?

"The right estimation for this project." The boss' words were still on Carmen's mind as she walked down the hallway to her desk. This was not Carmen's first project, but until now she had managed to give fair estimates on her projects because they were small and somehow similar. Take this system we already installed here, then install it there. Create a customized copy of that other system for a new client.

But this was a different kind of beast. New technologies, new client, new domain, new product. Until now, Carmen had just looked at how long things took in the past, then added some contingency buffer that could be accepted by the client. So far, so good. Of course, she also knew that keeping estimates 'good' during the project involved a strong management of the project. She was known to be very kind and empathic with clients, but 110% uncompromising when it came to scope creep and project changes.

"Carmen delivers." That was the usual comment of managers when they talked about her, and she liked it.

Now, her reputation had put her in a tight spot. For the Big Fish project there was no useful historic information she could rely on, and still she was asked to give 'the right estimate.'

Anyway, what was the 'right' estimate, or the 'wrong' estimate? Was this a game of guessing?

Carmen sat silently for some minutes trying to assimilate it all. One thing she knew: once she would cast a number, an estimate, it wouldn't matter if it was 'right' or 'wrong'. That number would instantly become known as 'the plan' and for the following months she would be measured against it. But on the other hand, if that number was 'too large' the company might lose the contract and she would be held accountable.

Carmen opened a drawer and pulled her old project management books in search of something she hoped would help her find the 'right estimate'. If she failed to produce the right estimate, at least she wanted to have the opportunity to say "hey, I did it by the book! Literally!"

What others are saying about NoEstimates, The Book

"The way this book was written helped me not only to understand the main ideas behind #NoEstimates but to internalize the concept with the real examples behind the story." – *Hermes Ojeda Ruiz, LogicalBricks*

"Finally! The most meaningful text book about project management and governance! This piece of project fiction will become the real "don't panic" guide to project management!" – *Tapio Järvenpää, Beer Executive Officer, Hops United OÜ*

"Anyone interested in deliver their projects on time should read this book. It´ll make you forget the idea that you can predict the future with a crystal ball. *Instead of that you´ll learn to use your data and be more effective. Take of red pill!" – Emiliano Sutil, Project Manager al Xeridia S.L*

"You're doing it wrong. Estimates that is, and #NoEstimates tells you why. Most importantly, however, the book tells you what you should be doing instead. The compelling mix of theory with story means you'll quickly learn the lessons and start benefiting from the book. Read, think and apply #NoEstimates today." – *Dr. Bruce Shcarlau, Senior Teaching Fellow at University of Aberdeen*

"I was sold on this book after reading the table of contents of the first chapter." – Nick Mein, Engineering Manager, Trimble Navigation.

"Do the most important thing is not a startup thing only! In my 15 years of experience with software development I've seen a lot of patterns of projects failing. I've experienced myself every example described in #NoEstimates. I've worked for a startup, where it is obvious what's important without estimating. With the help of the #NoEstimates book I can do the same within a multinational corporate organization*." – Jeroen Knoops, Software Developer*

"Trying to be more predictable by working with estimates is a seductively simple idea. After decades of widespread use of this idea, actual results are now very clear. Estimation has instead caused reduced predictability, lowered quality and unhappy customers. Read this book to start exploring some more successful strategies!" – *Henrik Berglund, Agile Organizational Coach, Cedur AB*

"A must read for all of us that has discovered that it's in the doing of the work that we discover the work that we must do." – Anders Lundsgård, Senior Engineer and Solution Architect, Sweden

Purchase the book at
http://oikosofySeries.com/noestimates-book-order

Buy Now!