OIKOSOFY SERIES

oikosofyseries.com

ESTIMATES

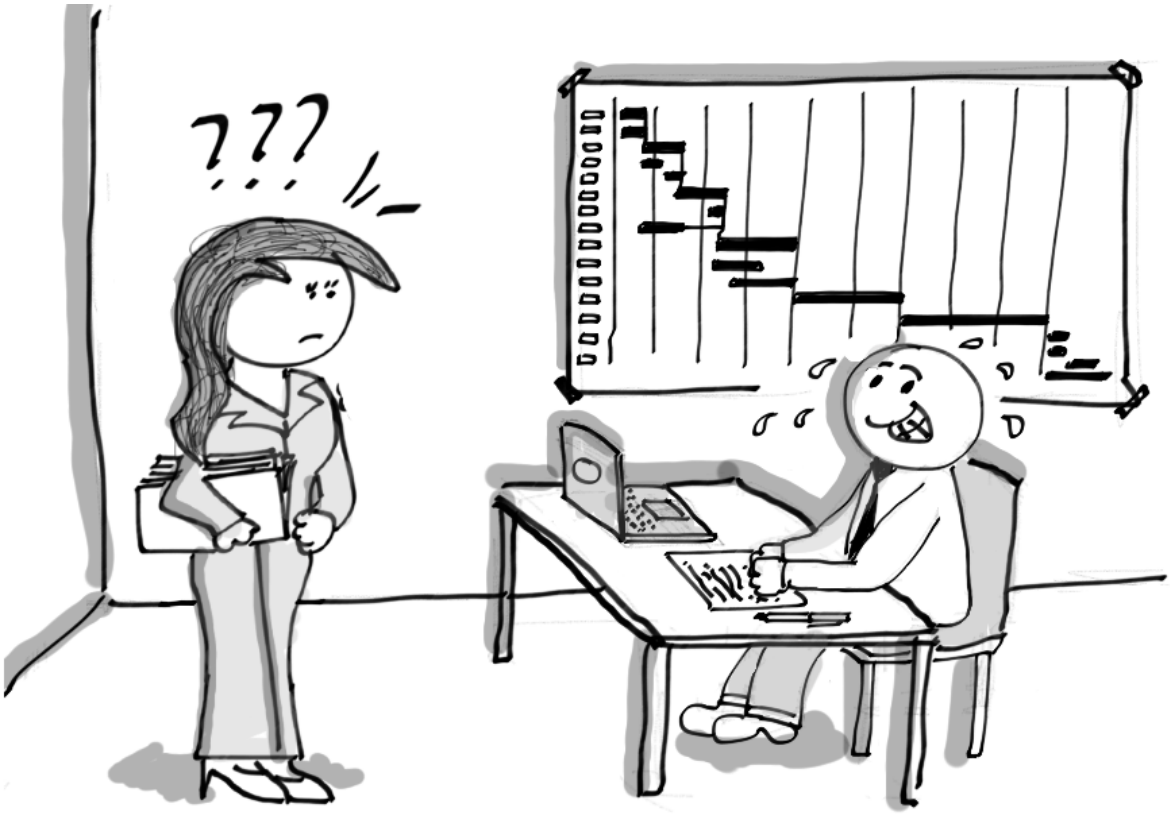
How to measure project progress without estimating

Vasco Duarte

Chapter 1 - The Problem With Estimates

Carmen comes in to the office; her boss – unusually – calls her into his room. Normally he would greet her, invite her for coffee and have a friendly one-on-one chat. Carmen appreciates her boss. She has learned a lot from him. But she is nervous. What would her boss want that requires them to meet behind closed doors? What is so secretive that can't be overheard by anyone else in the office.

- Carmen, we have a serious project coming in. This is potentially the largest software project our company has won. This project alone could make us profitable - finally! 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 client coming in - the Big Fish they named it, because it was so secretive – but she was not expecting to be assigned to this project. After all she still had the

Telemark project going on. "Something fishy is going on" - she thought to herself.

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

- Yes! This is a surprise deal, I only just yesterday heard about it. Our CEO was able to find a way for us to be able to bid on this project. – Carmen's boss interrupted – It can be huge!

- Errr, OK. What should I know about this before I accept the assignment? – Carmen was unsure. Her boss seemed almost too eager to get her to say yes.

- Well, basically nothing – he continued. It is a regular software project, a bit bigger than the others you've worked on, but basically similar to that Telemark project you are just finishing now. The key different is that this time, it is a bigger and more important client.

- How much bigger are we talking about? – Carmen asked.

- Well… – her boss was not being forthcoming. "What could be the reason?" she thought.

- Is that 2 times bigger than Telemark? – She asked. But her boss did not say anything – Five times bigger? – still no comment – Ten times bigger?

- Actually, it is a public sector project, so I guess you could say that it is about 347 times bigger customer than Telemark…. – he finally admitted

- Oh… – Carmen was speechless. This could potentially be the largest software project in the country, ever! And she was being lined up to manage that project.

- And we need to present a bid by end of next week! – Her boss looked excited. So get your team on it. We need to close this deal Carmen, it can change our company's future and save many jobs! We need submit our proposal soon!

Do Estimates work?

If you were Carmen you would probably feel nervous. I know I would. Estimating a software project is never easy, but the situation gets even worse when you have to do it quickly, without understanding the context or the requirements well enough. Add to that the pressure 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: "How can I win a bid if I don't estimate?", "My customers will not award me the contract unless I give them an estimate!"

Fair questions, but before tackling those let's tackle the real question here. A key question for Carmen and for you is: Do estimates work?

Some researchers have already – and for our benefit – decided what a "good" estimate is. They defined in 1986[1] a good estimation approach would provide estimates "within 25% of the actual result, 75% of the time".

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

What this means is that, for all the tasks/work packages you estimate, you should be within 25% of the original estimate for 75% of those tasks/work packages. Additionally, this definition also requires that all the projects we see in our industry, or in your company follow this rules: be within 25% of actuals, 75% of the time. Let's examine if this is the case in the IT industry.

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

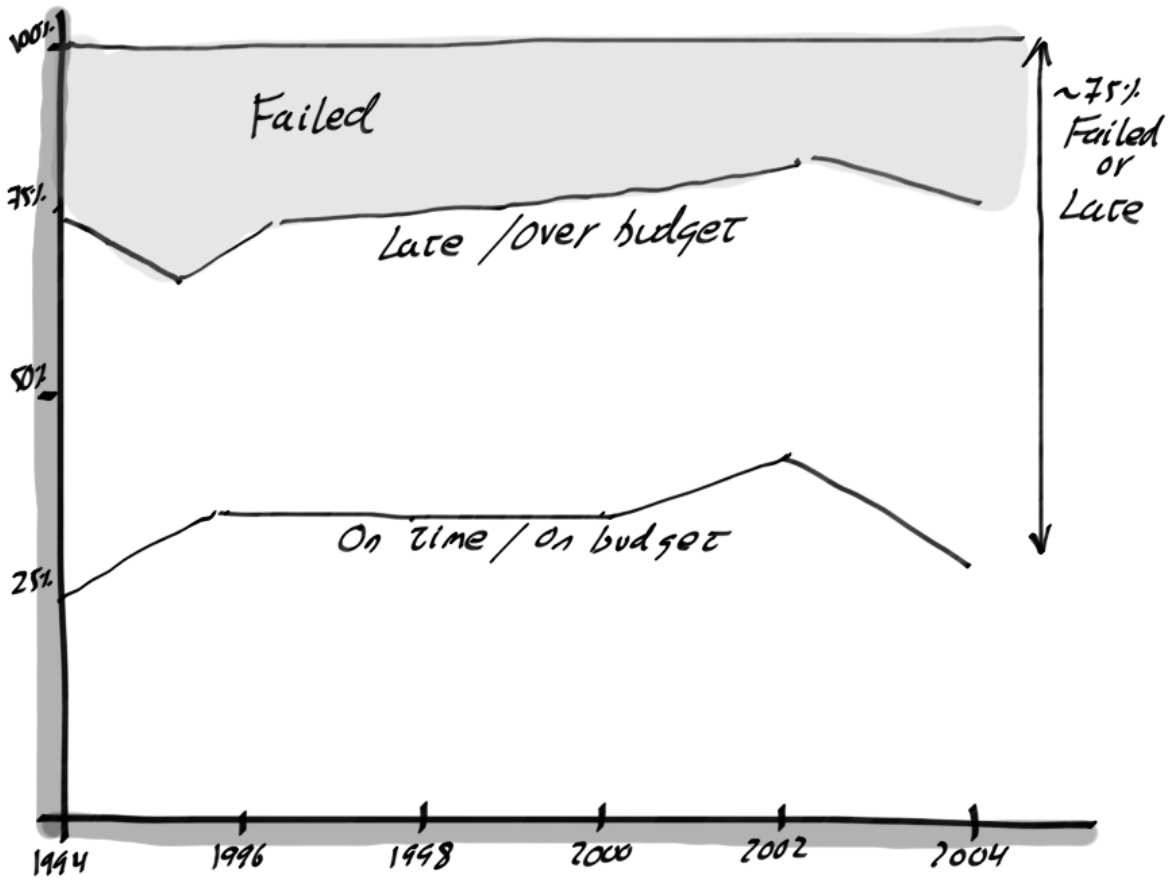


Figure 1 - Graph adapted from Steve McConnell's book, Software Estimation: Demistifying the black art

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

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

But let's look at another angle. Even if you agree that being within 25% of the original estimate is a good result. What happens with the outliers, the tasks and projects that exceed the original estimates by so much that they may jeopardize the whole project or portfolio? You may think that your projects do not have such tasks. Bad news…

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

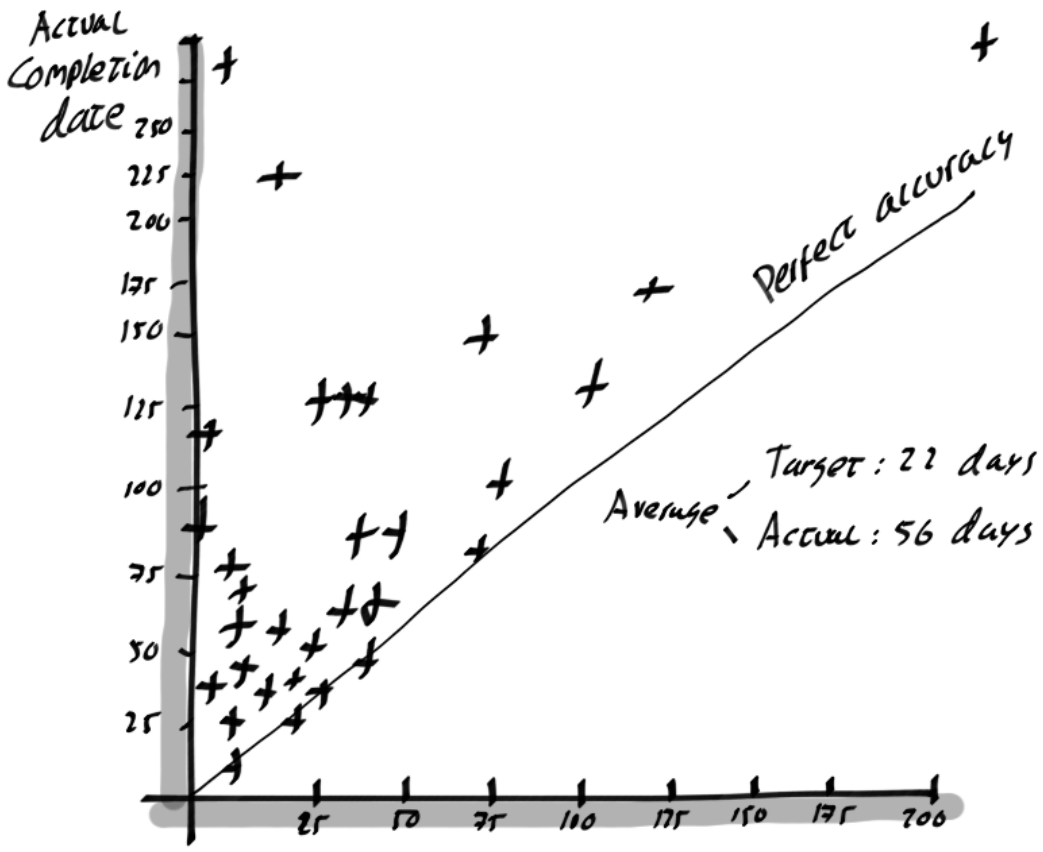


Figure 2 - Graph adapted from Steve McConnell's book: Software Estimation, Demystifying the Black Art.

Look at the project on the top-left of this chart. This was a project estimated to last 7 days, but ended up taking more than 250 days to complete. From 1 week to nearly 9 months! This was 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. Here's one example (http://www.theregister.co.uk/2006/09/29/accenture_nhs_penalty/) that could have led to Billions of Dollars in penalties. An outlier for sure, but do you want to bet your company on one project if the possible result is obliteration? I didn't think so!

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

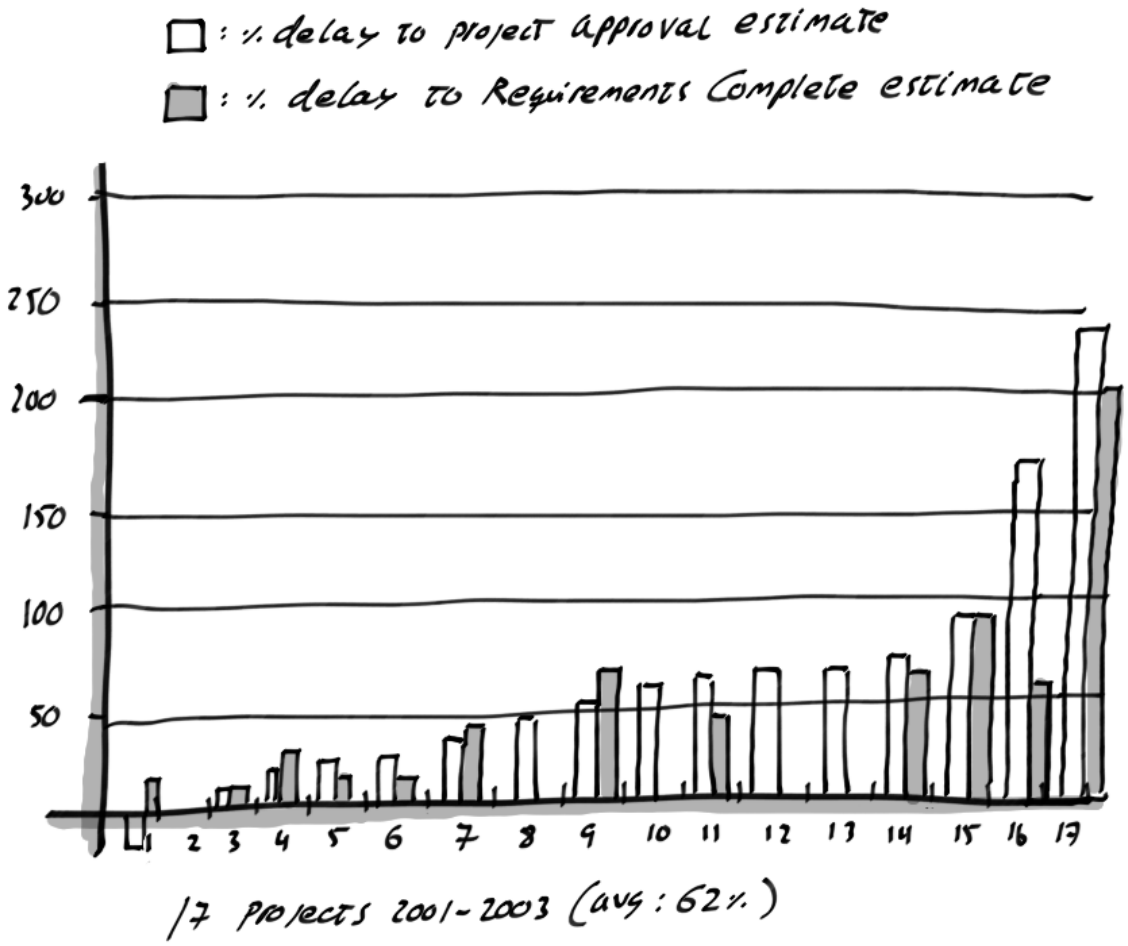


Figure 3 Adapted from data collected in one company between 2001 and 2003 for 17 different projects

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 company on a work-method (estimates) that delivers 60% to 80% of the projects late or over budget? This is the question that Carmen was asking herself in the scene above. Can she, in all honesty, make her company's future depend on a method that has that large 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. – Her boss finalized.

- Thank you, 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 created. There are three clear, intuitive, and proven reasons 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, *Gödel, Escher, Bach: An Eternal Golden Braid*[3]

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

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

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

Parkinson's Law

Parkinson observed that bureaucracies seem to expand and therefore create more work for themselves. He was commenting on the British colonial bureaucracy, but the rule has been picked up later by Alessandro Natta in Italy and Mikhail Gorbachev in the Soviet Union who famously stated that "Parkinson's Law works everywhere[4]".

Indeed, it works everywhere; even in software development environments where the modern formulation of Parkinson's Law is "work always expands 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 no work will ever be completed before the time allocated to it. Which in turn means that if you have one task that is delayed, you should consider all subsequent and dependent tasks to be late. Therefore, having one task late is potentially enough to derail the whole project. How about that for a shot in the dark?

---

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

[4] O'Sullivan, John (June 2008). "Margaret Thatcher: A Legacy of Freedom". Imprimis (Hillsdale College) 37 (6): 6.

Accidental complication

Another reason why estimations can't work goes by the name of Accidental complication. J.B. Rainsberger introduced this concept at Oredev 2013, in a lightning talk of which you can see the full video here: http://vimeo.com/79106557.

But the gist of the argument is this: every feature or functionality added to an existing project has two cost elements (what you 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. 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 and Accidental complication:

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

In software, often estimates of cost are done by analogy. If Feature A and B are similar, then if Feature A took 3 weeks, Feature B will take 3 weeks. But then J.B. 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 - and I agree - 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 - which is the most common estimation approach in teams that use, for example, Story Point estimation. 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.

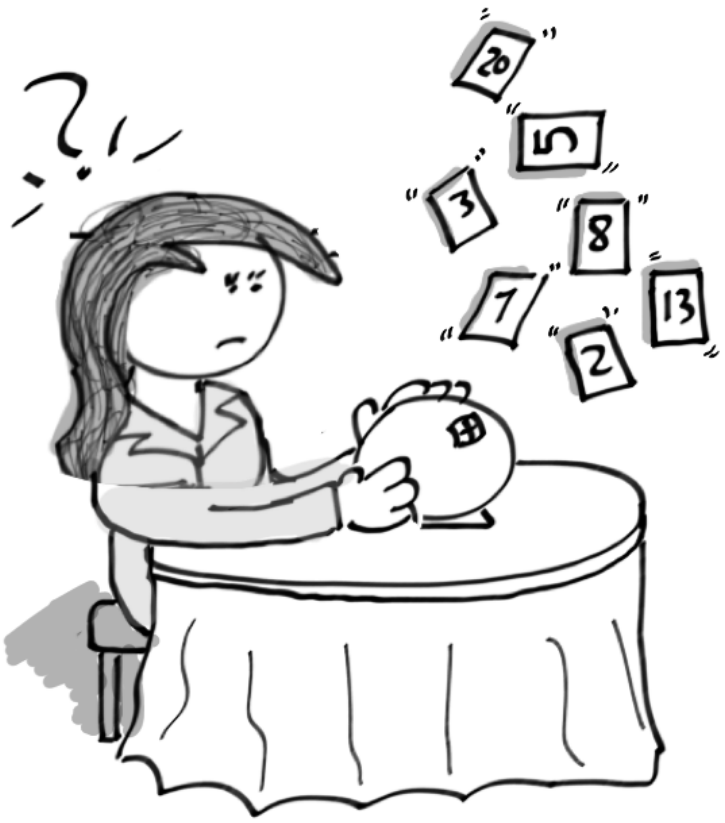
More than meets the eye: why estimates are actively harmful for your project and your company

- 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.

- Of course! – she answered.



I would not like to be in Carmen's shoes. What is she to 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[5] used to say 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[6] where the

[5] 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.
[6] For details on the Enron scandal, visit http://en.wikipedia.org/wiki/Enron_scandal

"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:

- ***Internal politics***: when deadlines are set because top management wants something done by a certain date, with no possibility to reduce the scope of that release. In these cases the HIPPO (Highest Paid Person's Opinion) overrules any potentially "good" estimate created by the team.
- ***Estimate bargaining***: Once a team has created the estimate and defined a possible release date, they start suffering from pressure from stakeholders to change their estimates or be called "slow" or "inefficient".
- ***Blame shifting***: when certain people create the estimates for the work (e.g. experts), and then other people (e.g. the project team) is blamed for not meeting those estimates.
- ***Late changes***: 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, but they are enough to illustrate how estimates can be used and abused for other purposes 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 complexity. How complex dependencies in software projects hide the really big risks.

Software projects present challenges that are unique to the software development domain. But not everybody agrees with me. 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.

Additionally, 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[7].

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 type of complex causality: non-linear causality, i.e. when

[7] For more details visit: http://www.digitopoly.org/2014/02/26/apples-goto-fail-is-pretty-interesting-and-worth-reading-about/

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 this kind of small problems with disproportionally large impacts. And there's a very good reason for that: they are rare. They are so rare as to be considered outliers. The problem is: in software, an outlier can potentially 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:

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 80K lines of code won't fit into a 20-page contract.

Do you have some more quotes to share? Send them over, I would like to collect them. Email me at vasco.duarte@oikosofy.com

Change requests, change everything

And then there were changes. You have been there before. You work hard to try and plan a project to 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 at the rate of an avalanche.

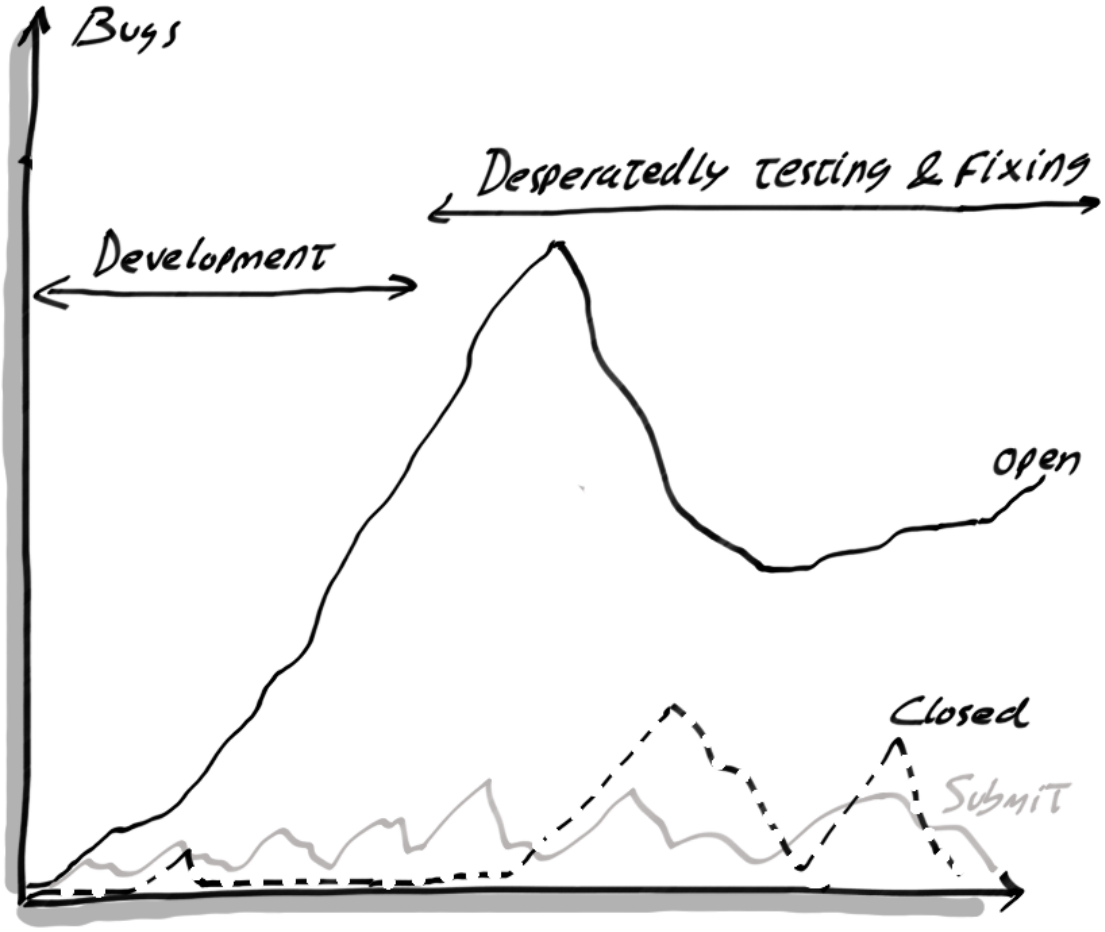


Figure 4 Adaptation from a medium-size project. Depicted in this graph is the curve of open errors or defects during the project. Includes indication of development and testing phases

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 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.

Changes have side-effects, as you discover problems you discover the consequences go beyond what some call “bugs”, you discover blind-spots. 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 an three more houses just to get the toilet working”.

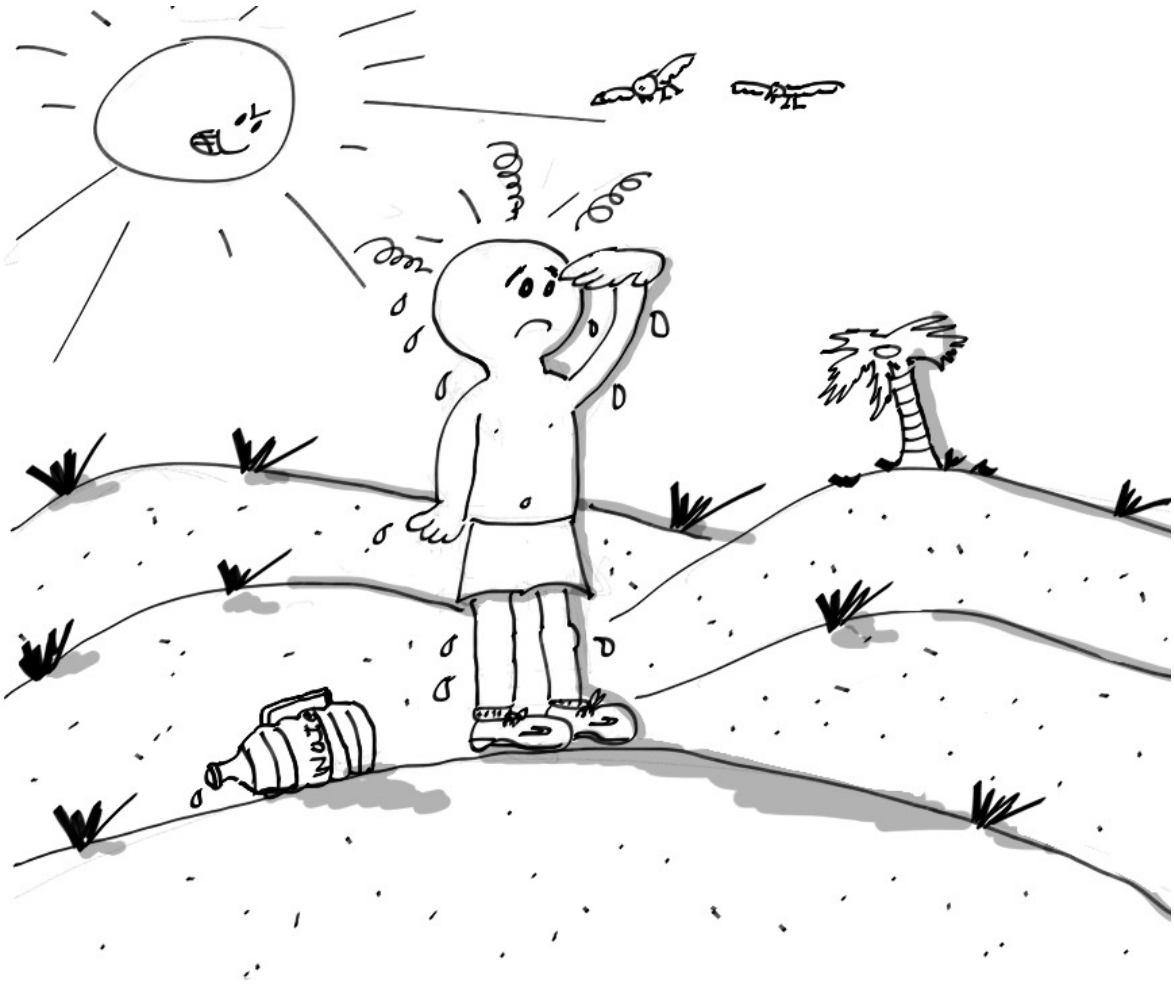
Can you predict all the significant changes you will get, and the impact of those changes? If you answer yes please send me your CV, I may want to work with you in some of the largest projects on the planet. But you will probably answer “no”, just like many people do when asked the same question. The 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[8].

The idea of the Death-March project 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've finished with all your 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.



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").

[8] Article on the incidence of burn out and stress related illnesses in IT http://www.itburnout.org/2013/03/27/gfi-software-survey-most-it-admins-considering-quitting-due-to-stress/

In this context, the estimation process becomes a battleground between the teams and the buyers of that project. The teams want to survive at the end, and the buyers 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. Where 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 Sysyphus[9]: pushing a boulder up a hill that will 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 an 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: *"Give 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.
- 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".

But before exploring how a #NoEstimates practitioner would predict an end date for a project or how much work can be done, let's review, in the next chapter, some of what the software industry today considers "best practice" methods for software estimation.

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

Do you want to get the whole book for free?

Sign-up at NoEstimatesBook.com

Do you have questions about the content in the book?

Email me at: Vasco.Duarte@Oikosofy.com

Share your thoughts with the #NoEstimatesBook hashtag on twitter