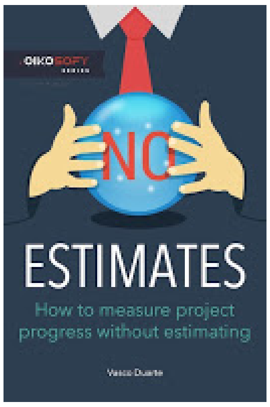
Read the brilliant #NoEstimates book by Vasco Duarte – a must for being up to date regarding agility

ontheagilepath.net/2015/09/read-the-brilliant-noestimates-book-by-vasco-duarte-a-must-for-being-up-to-date-regarding-agility.html

By Sebastian Radics



Source:http://noestimatesbook.com/

This week I read Vasco Duarte's brilliant new book about #NoEstimates and I just have to write a post advertising it.

Finally there is a ground setting summary about the #NoEstimates movement and I can highly recommend reading it.

For me the combination of a business novel style (like done in the Phoenix Project, The goal or Five Dysfunctions of a team) – the story about Carmen the project manager – and the explaining fact based surrounding that support Carmens development throughout the book is a great way to tell the story about #NoEstimates.

It took just some hours to read it and helped me a lot to get an even more clear picture why applying #NoEstimates is the way to continue in agile projects.

And you need to see the beautiful visualizations by Angel Medinilla (I still remember his great Aikido session and amazing sketchnotes at #Dare13).

Some highlights as a teasers for you

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 of triple it?

#NoEstimates: estimates do not directly add value to your process … can we find ways to reduce the estimation process or even stop it where possible.

Forecasting … to calculate or predict (some future event or condition) usually as a result of study and analysis of available pertinent data … we use actual data to make calculations instead of using experience and other information to guess at something.

…you don’t want to triple plans, estimates, reports and proposals…

…finally dropped the estimation-based conversations to focus on throughput, cadence and flow…

…#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.

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

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

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

Accidential complication (by J.B. Rainsberger): applied to #NoEstimates … 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.

…Work spends considerable more time idle and blocked by system constraints than progressing towards completion… It’s these unknowable system delays that make forecasting software projects exceptionally difficult…

The harming effect when using estimations: internal politics, estimate bargaining, blame shifting, late changes.

…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… in software, an outlier may completely destroy all the effort that went into creating credible, solid estimates.

Can you predict all the significant changes you will get and the impact of those changes … “building software is like when you try to unclog the pipes in your bathroom and end up to build three more houses just to get the toilet working again”

In software there are many factors affecting performance on the job, and those factors are not predictable nor do they exhibit predictable variation… our intellectual performance is the key determining factor of our productivity and many factors affect us… when you base estimates solely on expert estimation you fail to account for many aspects that affect directly the performance of the project … then you use data from the project to update your forecast, you are factoring in all the aspects of the current project and will be able to detect when the project performance changes.

You need to ask: “given the rate of progress so far, and the amount of work still left, when will the project end?” or similar “given the rate of progress, how much of the work can be finalized by date X?”

Estimates … to give or form a general idea about the value, size or cost of something … are guesses, not facts … as soon as the human mind understands this number it will likely tend to stick to it and give it the category of a fact, in an interesting form of grasping and clinging…

By asking people to reduce their standards or stay over time, you destroy their motivation because of a commitment made based on an estimate.

The start of a project is the worst time to estimate its duration or cost.

…when every task has some specific mini-buffer (padding), Parkinson’s law kicks in and all tasks would expand to consume the available buffer…

…once you know which is the one single most important thing in the universe for your company right now, there is little value in knowing how much is it going to cost when compared to other items in the backlog…

…if you are able to perform this series of strategically critical experiments so quickly, then spending time trying to estimate the length or cost of each experiment is pointless…

…Identifying which features are the ones that deliver most value is therefore more important than finding which of them are small and which are big…

…estimates … cannot give you … visibility to progress and actionable information…

project driven development anti-pattern… other metrics such as quality, product market fit, speed of delivery end up being ignored in favor of being on time and budget.

…make design commitments as late as possible it the project … you have to wait until the last responsible moment; if you move too early, your enemy will spot your movement and neutralize it; but if you wait too much, your opponent will strike you…

heijunka … to reduce variability in the work where possible … by having small user stories … will help you reduce variability in task duration as well as help you identify the unexpected variability early on.

8 possible sources of variability for software teams: technology, domain or product, team composition, user and client, workload and/or focus factor, market and competitors, dependencies and specialization, waiting for availability

- Stabilize your teams and avoid constantly changing people from one team or project to another.
- Define clear priorities, reduce dependencies and allow people to work on one thing at a time, finishing it before starting a new work item

- Build quality into the whole production process, do not try to inspect quality in at the end of the line…ensure that no defects are passed down to the next step in the line
- enforce cross functional groups and generalizing specialist teams
- Standardize and automate where possible
- Protect the team from outside interruptions
- Freeze scope inside iterations and allow the team to work on delivering the most important work without context switching that costs them time and takes focus away from the most important work

…when you are ready to respond quickly, no matter what your competitors come up with, you will be able to respond and adapt.
Prioritization comes before estimation.
As a consequence of estimation … the estimations becomes the real value for a feature … because it is so easy to do first what is considered "cheaper".
Product owner … divide any project into stories of only 3 relatively small sizes … small stories are finished on one to three days, medium stories are finished on three to six days and big stories are finished in six to twelve days …

- nothing bigger than Big
- sizes in your backlog come randomly (e.g not all small stories in the beginning)
- the statistical distribution of the sizes would be constant and known over time

…the ability to slice work into "small enough" work items or stories will give you a high degree of throughput predictability…
…a better way to manage your projects is to acknowledge uncertainty, embrace change and fix the hardest constraints (time, people, resources) while letting uncertainty be managed through a variable scope.
"The best way to achieve predictable software development outcomes is to start early, learn constantly, commit late and deliver fast"
…If we batch the fixing of bugs to the end of the project – which we have to do if we take the layered approach – we will not know the real progress until we start fixing those bugs and very likely later than that…
WORKING SOFTWARE IS THE PRIMARY MEASURE OF PROGRESS.
Developer done … I've coded in what I think is the meaning of the requirement you gave me…

- verify that the developer has the same understanding as someone else (e.g. Tester, PO)
- ask the end customer if what is implemented really solves the problem that it was intended to solve

Progress in software is measured by RUNNING TESTED SOFTWARE (stories) … RTS is the only metric that reliably describes the progress in a software project
#NoEstimates slicing principles:

- there should be no huge stories (bigger than half a sprint)
- several independent stories should fit into a sprint … six to twelve stories delivered in a 2 week period is a good rule of thumb

INVEST for stories – adapted…

- E – essential (instead of estimated) … meaning that every story is absolutely required for the product to be viable
- S – small (instead of specific) … small stories are well understood by the team and can be implemented in a short time frame

…I advocate that stories should be between 0,5 and 1 man days of effort…
Plotting progress based on RTS is giving a clear and verifiable progress information and actionable information.
Figure out:

- What is the most important value to be delivered by the project from the customer's perspective?
- What is the amount of work that needs to be completed in number of stories to be delivered?
- When should the next delivery of the project happen?

by asking:

- What are your intermediary deliveries going to be used for?
- When does the project need to go live with the first release?
- What does the customer expect to accomplish with that first release?
- How many and which RTS do you need to deliver until that first release?
- How many RTS have you successfully delivered during the last 2 months (the length of the project until then)?

HAVING A CONSISTENT RATE OF PROGRESS is more important than estimating a project.
Slicing of stories:

- Along functional dependencies
- to separate quick from time intensive functionality
- along user/role separation
- …

…early in the project your top priority is not to ship something meaningful to your customer, but to obtain information on capacity, throughput and backlog size…
…in my research I've normally used the progress data from 3 to 5 iterations in order to define the initial progress rate…
Features (an aggregation of stories and a valuable bigger unit for the customer) – should be given a mandatory, maximum calendar duration.
How many user stories can a team deliver on an average week (story velocity)? How many features can our project deliver on an average week (feature velocity)?
The reason limiting work to a fixed time works is simple: it forces you to increase the frequency at which you collect feedback and evaluate progress.

- use multiple levels of granularity to establish flexible requirements
- use historical data to assess progress and forecast future delivery
- cost is a multiplier of time; the only real variable the project team can control actively is value (in the form of scope)

Step by step toward #NoEstimates:

1. Move to story points
2. Stop estimating tasks
3. Limit calendar duration of Features and Stories
4. If you already are using story points remove some planning poker options
5. Build histograms to track average duration for stories and features
6. Use average cycle times for stories
7. Finally work with stories as if they all had the same duration

…In fact, slicing stories is probably the most effective tool to manage scope and very few projects are using this approach effectively today … STORY SLICING IS THE LOST ART OF AGILE SOFTWARE DEVELOPMENT
By forecasting and showing progress very early on you are bringing the scope discussion to the first few days of the project … very few dependencies are created, unlike at the end of a project where removing some functionality may be harder than adding it in the first place…
…as we know more about the project (a feature or story) we question the need for that work and if possible remove that work…
…as you describe the scope of your project in the form of a matrix (STORY MAP), instead of a list you are able to see possible options for reducing scope much more clearly than if you only consider the list of features…
…when we slice features we create options that allow us to pro-actively manage the scope of a project… by creating different implementation options that are often implicit and non negotiable when we have larger features in the backlog…
Rolling wave forecasting allows you to adapt your plans constantly… your goal should be to make changes to the plans easier … the simple the planning process, the faster and more easily your projects will be able to accommodate the inevitable changes.
UUPS – the teasers list just became a little longer … I hope it inspired you to start reading the book and maybe share your opinions e.g. via a comment or join the discussion under #NoEstimates or #NoEstimatesBook.

Thank you Vasco Duarte for your efforts in writing this book. It’s really a great help and will bring a lot more momentum in the #NoEstimates movement.

Further readings

Are your still estimating, apply #NoEstimates and simplify your process – a practical guide

- Lean from the trenches by Henrik Kniberg (or my summary)
- The No Estimates Principle by Vasco Duarte (looking forward for his great book about #NoEstimates coming soon)
- #NoEstimates on Twitter