Agile Project Estimation and Planning

InfoQueue
eMag Issue 17 - August 2014

Interview with Eduardo Miranda about Estimating and Planning Agile Projects

Eduardo Miranda, associate professor at the Master of Software Engineering program at Carnegie Mellon University explains the need for planning in agile projects, and describes various planning techniques that can be used with agile. He also looks on the impact of agile on project management offices and on the role of project managers in agile projects.

PAGE 4

Contents

A Letter from the Editor

Shane Hastie is the Chief Knowledge Engineer for Software Education (www.softed.com) a training and consulting company working in Australia, New Zealand and around the world. Since first using XP in 2000 Shane's been passionate about helping organisations and teams adopt Agile practices. Shane leads Software Education's Agile Practice, offering training, consulting, mentoring and support for organisations and teams working to improve their project outcomes.In 2011 Shane was elected as a Director of the Agile Alliance (www.agilealliance.org)

Estimation is often considered to be a black art practiced by magicians using strange rituals. It is one of the most controversial of activities in Agile projects – some maintain that even trying to estimate agile development is futile at best and dangerous at worst.

There are indeed times when estimating is unnecessary and/or pointless. If the work is novel, nothing like it has been attempted before, the skills of the team (or who the team will be) are unknown or the work just has to be done (either the value to be derived from having the product is compelling or it is a compliance or survival project) then estimating may be a waste of time and effort.

The more common reality is that there is a need to provide at some sort of estimate of the likely time and cost needed to build the product. Business decisions need to be made regarding which initiatives to fund and the allocation of funds and people to do the work. However there are lots of risks and mistakes associated with estimation, and there are a number of alternate approaches which can be used.

When putting together this ebook we selected articles which present ways of coming up with estimates as well as some that argue for alternate approaches.

Ben Linders spoke to Eduardo Miranda about estimating and planning approaches in agile projects, why the traditional estimating and planning approaches don't work and ways to be predictable using agile methods.

Jay Fields presents a number of different ways to size and estimate user stories in agile projects, presenting the pros and cons of a number of different techniques.

David Morris discusses why estimating matters, some of the advantages and disadvantages of estimating, presents some approaches and examines the #noestimates debate.

Alex Adamopoulos and Paul Dolman-Darrall address the use or not of numbers for prioritization of features or user stories.

In Neil Killick's presentation at Agile Australia 2013 he proposed ways to reduce risk and uncertainty, calculate a product's price, determine delivery dates and roadmap, do Scrum and XP without using estimates.

Finally Jutta Eckstein argues for a different approach based on beyond budgeting which is designed to increase flexibility and maximise business value while avoiding unwanted surprises.

We hope these articles will help you address your estimating challenges and provide food for thought with some ideas of different ways to approach the topic.

Interview with Eduardo Miranda About Estimating and Planning Agile Projects

by *Ben Linders*

The usage of agile planning practices – like product backlogs, planning games, and stand-ups – impacts the planning of projects and portfolios in project-management offices. In an interview with InfoQ, Eduardo Miranda, an associate teaching professor in the Institute for Software Research at Carnegie Mellon University, explained the need for planning in agile projects and described various planning techniques that can be used with agile.

He also looked on the impact of agile on project-management offices and on the role of project managers in agile projects.

InfoQ: Eduardo, could you briefly introduce yourself to the readers of InfoQ?

Eduardo: Thanks for this opportunity and the interest in my work. I am a software professional with over 20 years of experience in the development and management of software and process-improvement programs. I spent my last 10 years in industry, working for Ericsson in Canada, and now I am an associate professor in the Master of Software Engineering program at Carnegie Mellon University. I have published numerous articles on software development, estimation and planning and a book on project-management offices, Running the Successful Hi-Tech Project Office, which was published by Artech in 2003.

InfoQ: Newer methods in software development like agile and lean are changing the way that planning is done. Some people even question if there is still a need for a plan. Are plans still useful?

Eduardo: Plans serve several purposes. First, they help us think how we are going to approach the work before we start it. Second, they communicate to the stakeholders to expect the project outcomes so they can plan their own activities. Third, they help coordinate the work among team members.

Iteration and daily meetings fulfill this last function but not the first two. So yes, plans are still useful. The problem, I think, is many people confuse a plan with an activity network or a task-precedence diagram, which are just tiny fractions of the planning work.

InfoQ: Are there planning techniques that agile teams can use in their planning games and stand-ups. Could you name some, and explain how they can help the teams?

Eduardo: There are many techniques that can be used. Milestone planning is one of them. Milestone planning is planning in terms of intermediate and end goals to be accomplished by the project. This is the plan that could be used to communicate with the project sponsor and other stakeholders. It will specify when they should expect to receive some functionality or provide information and resources to the team. This plan is typically very stable, can be built early on the project, and will make visible the impact of changes. “Making visible the impact of changes” doesn’t mean “not embracing change”, it simply means that when you change something, other parts will move, and you want to be sure your customer understands the choices he or she is making.

Another one is the paired-comparison technique for estimation. The idea is the same as in the planning poker to establish the relative size of user stories or features. Unlike planning poker, paired comparisons force you to compare one user story to several others. This triangulation of estimates reduces the possibility of getting the numbers wrong and is later used to compute a consistency index and average sizes. In several experiments I have performed in industry and in the classroom, paired comparisons provide more consistent estimation at the expense of more questions. The planning-comparison method is described in a number of articles, one by Martin Shepperd and Michelle Cartwright – “Predicting with Sparse Data”– and another by me: “Improving Subjective Estimations Using Paired Comparisons”.

There is also a tool that implements the method that can be downloaded: the Paired Comparison Estimation Tool. In both articles, the idea of using comparisons, which was first proposed by Thurstone in 1927 to measure social values, is the same but the underlying math is different. In any case, the math part should be isolated from the estimators and is only relevant to those wishing to develop a tool to support the method.

InfoQ: I know milestone planning from projects that didn’t work agile. When we adopted agile at Ericsson, managing milestones as we had been doing with iterative projects became difficult and didn’t really help the agile teams. Can you give examples of how to use milestones with agile, product backlogs, and changing priorities that impact scope?

Eduardo: As you know, I also worked for Ericsson. The milestone plans we prepared there were task-oriented and dates at which they were due calculated using an activity network that started the first day of the project. The milestones I am talking about here are different. I will try to summarize the idea in five sentences but for the interested reader I recommend an article and a book, both written by Erling Andersen. The article is “Warning: activity planning is hazardous to your project’s health” and the book is Goal Directed Project Management. The difference between what Andersen proposes and what we did at Ericsson is that milestones in his approach correspond to things that are relevant to the sponsor and the team; they are chosen to represent the state of commitment between them. For example, let’s say the sponsor needs to provide a special device for the team to prototype a GUI on it – this will be a milestone.

Following with the same example, let’s say the sponsor needs to arrange a marketing campaign around the new GUI. He will need to have a demonstration – not the complete GUI - finished before the end of the project so he can incorporate some images in his campaign. These two dates cannot be in total flux. The first one because the team cannot complete the prototype until the device is delivered by the sponsor; the second because the people running the campaign have lead times for contracting publicity spots, etc. In this example, notice that the relationship between the first and the second milestone is a finish-to-finish relation: it says the prototype cannot be completed until the device is delivered. It does not say we cannot start working on the prototype until we receive the device. This is a fundamental difference with what we used to do at Ericsson.

Tasks are planned from the milestones backward. So, for example, when you plan an iteration, the prioritization will have to take in consideration this global view of the project and not only the immediate concerns. If you need to change the milestones, so be it, but you need to be aware of how that could impact other things down the road. The first milestone is what I call a “soft milestone”, a milestone that is under control of the team. If the delivery of the device is delayed, it might have an impact on other commitments the team has made or increase the risk but the world will still be there the day after. The second milestone is of a different nature; it is a hard milestone. It is imposed externally by the advertising agency running the campaign. If the images are not submitted, the campaign might not be launched, and that bring us to the next point: time-boxing and commitments.

InfoQ: Ok, let's talk about time-boxing. Agile teams use time-boxed sprints to prioritize schedule over the deliverables. It helps teams to deliver at regular intervals. Do you think that time-boxing and sprinting are also effective ways to plan the work into projects or releases?

Eduardo: Fixed-length iterations are good to help the team keep the pace and receive timely and regular sponsor feedback but dividing the total available time in two or four-week chunks is not, by itself, something I would value as a sponsor. Time boxes without agreed outcomes and decisions made two weeks in advance are at best an expression of good will on the part of the team. If, as a sponsor, I am counting on the software to launch a marketing campaign, I need something more solid than "The fourth iteration will be completed by the third week of March." I need to know what the team can guarantee I will have by that week in terms of functionality, level of service, support, etc.

InfoQ: Isn't this conflicting with agile concepts where the product owner and not the team takes responsibility for the scope to be delivered? And where the team only tries to deliver something in one sprint and doesn't even commit to delivery, which is what the latest version of the scrum guide of Beck and Schwaber states?

Eduardo: I do not subscribe to this concept. I think we can do much better than discovering what needs to be done in a project two weeks at a time. Sponsors and teams need to have an idea of what they will be doing and by when. Do they need to know if they will be unit testing the XYZ class on April 21 at 9:30 a.m.? Definitely not. Do they need to know that they will receive the special device of the example the second week of April if they are going to have the prototype ready by the first week of March? Definitely yes.

The fact that you might not know when a given task will be performed or if a particular feature of a deliverable will be included or not does not mean that you don't know when the overall work leading to a deliverable will have to be performed nor when the deliverable is needed or expected. The owner can take responsibility, but the team must bring to bear its knowledge and experience to commit. That is the difference between a professional and an amateur.

InfoQ: Some agile teams use MoSCoW to prioritize their backlog. Product owners assume that a team can finish the must-haves and probably also the should-haves before the delivery deadline, but they sometimes don't know if it's possible. Is there something they can do to increase the certainty about the scope that will be ready on the deadline?

Eduardo: I have proposed two methods I have successfully applied in industry and academia based on the ideas of incremental development and the critical-chain project management developed by the late Elyahu Goldratt. I called the first approach SPID (statistically planned incremental deliveries, a bad name looking back at it) and the second "buffered MoSCoW rules". The ideas behind both approaches are the same, but the second requires only addition and subtraction of man-hours versus the aggregation of statistical distributions employed by the first.

The simplification is not free. It comes at the expense of the claims we can make about the likelihood of delivering a given functionality and the overestimation of safety. Documentation for both methods can be downloaded from my Web page, Eduardo Miranda Publications.

InfoQ: Getting better insight into the certainty when user stories will be finished sounds good. But customers, marketing, and sales are used to fixed deadlines with an agreed-upon scope. How can you deal with that?

Eduardo: The buffered MoSCoW rules and the SPID method that I described above can provide the certainty required. The members of the team estimate the uncertainty in terms of normal and worst-case development effort for each feature or user story and then ask the sponsor what things he must absolutely have by the end of the time box. Now..., you would only commit to those (must-have) things you can do in the allowed time under the worst-case scenario. That is how you guarantee you can deliver what you say you will deliver. (Of course, if the worst-case scenario is worse than you imagined, then you might still not be able to deliver.)

Once you have selected the things you can accomplish under the worst-case scenario, you will incorporate them into the plan using the normal-case effort. This will effectively create a white space within the time box you can fill now with should-have features by repeating the procedure. If during execution, worst comes to worst, you will push out the should-haves and use the space left for the must-have features. Because the total effort you have now corresponds to the sum of the worst-case scenarios, by definition you should be able to deliver on your commitment.

InfoQ: Many organizations use project-management offices (PMOs) to manage their project portfolio. When they want to adopt agile, how does this impact their PMO?

Eduardo: As you know, there are different types of PMOs. Some are in charge of enforcing a standard process, others are aggregators of information for the executive levels, and others are involved in the balancing of resources and the management of the project portfolio. Given the space constraint of this interview, I will concentrate in the last function: managing the project portfolio.

One of the big challenges of managing a project portfolio is to avoid the propagation of delays from one project to the next through the links created by using shared resources, i.e. the developers working in one project are not available to work on other. To do that, there are several things that can be done. The first is to separate projects that are amenable to time-boxing from those that are not and not using resources allocated to projects of the second class in those of the first class without considerable lead time. The second is to use resource countdowns and clear priority rules among projects to minimize multitasking and to let shared resources know in advance that they will be needed somewhere else soon so they and their teams can get ready.

InfoQ: Does this also mean that the role of the project manager will change when an organization adopts agile?

Eduardo: Definitely. In many cases, project managers, in the traditional sense of the word, are no longer required. Take, for example, the role of the project manager as pace keeper. In this role, the project manager helped the teamIntervieweece of work through a combination of organizational and personal power-using tools and techniques such as inspiration, status meetings, time-reporting, recognition, rewards, warnings, and sanctions. In agile projects, this monitoring and controlling function is replaced by the peer pressure implicit in the practice of daily meetings and is illustrated by two of the three questions – what did you do yesterday and what are you planning to do today? – to be answered in a scrum meeting. Another example would be the change of task allocation from a push to a pull mechanism and the collective ownership of backlog.

InfoQ: So project managers will not be planning projects in detail, and following up on the activities when they work with agile teams. How do they manage scope, time, and money in agile projects?

Eduardo: As I said before, the traditional role of the project manager as owner of the plan and task expediter does not exist in most agile approaches. In [The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework]() (April 2, 2012), Sutherland states that in a scrum project, the scope, cost,

ABOUT THE INTERVIEWEE

Dr. Eduardo Miranda is an associate professor at the Master of Software Engineering program at Carnegie Mellon University. Before joining Carnegie Mellon, Dr. Miranda worked for Ericsson where he was instrumental in implementing project-management offices (PMOs) and improving project-management and estimation practices. His work is reflected in the book Running the Successful Hi-Tech Project Office, published by Artech House in March 2003. Dr. Miranda is a certified Project Management Professional and a member of the PMI and the IEEE.

READ THIS ARTICLE ONLINE ON InfoQ

and completion date are “set during the project”, i.e. projects using the agile approach only commit to doing their best. According to the literature, what gets done is decided at the beginning of each iteration, with regards to time and money. There are two approaches. One is when you run out of time and money you are done – this would be the time-boxing approach. In the other approach, you keep going on until the sponsor says it is enough; this would be more a time and material approach.

User-Story Estimation Techniques

by *Jay Fields*

One of the great things about working as a consultant is the ability to try out many different ideas and to adapt your personal favorite process to include things that work. This article details user-story estimation techniques that I've found effective.

Powers of two

Originally, I estimated stories as 1, 2, 3, or 4, or as small, medium, large, or extra-large. It was always meant to be understood that a medium was twice the size of a small and a large was twice the size of a medium (and so on), but that never seemed to translate well when it came to planning. Then someone recommended that I try powers of two. Suddenly we were speaking a language that the business could understand. They knew that an 8 was significantly bigger than a 1.

I believe the sizes 1, 2, 4, and 8 are also much more appropriate. As stories get larger, they almost always contain more unknowns and risk. Scaling by powers of two emphasizes the risk associated with large stories.

Use four values

I was once on a project that started with 1, 2, 4, and 8 as their estimation values. After the first two estimation sessions, fewer than 5% of the stories were 1's and about 30% of the stories were 2's. The project manager decided to get rid of the 1 value because it made his life easier. An interesting thing happened at subsequent estimation meeting. Suddenly only 5% of the stories were 2's and many more stories had become 4's.

I don't think that the developers consciously changed their scale, but developers are conditioned to be skeptical. Few developers are willing to say with certainty that any given story will be as easy as the scale allows. After witnessing this type of behavior on a few different projects, I prefer a minimum of four point values. I also prefer a maximum of four point values. After all, it's nothing more than an estimate. If you try to give too much precision to an estimate you'll end up having to account for why you missed the mark. The idea is to get a rough idea, not a rigid plan to live off.

No averages or numbers not on the scale

Four values allow you to roughly estimate without spending unnecessary time focusing on precision. Sometimes a story feels larger than a 2 but smaller than a 4. The story should not be estimated as a 3. There's really no reason to use a 3. The story carries enough risk or unknowns that it is not a 2; therefore, it's very likely that it will actually be a 4. Using an average or off-scale number can briefly (and unnecessarily) confuse a team member or stakeholder. Also, in the big picture of the project, the occasional uncommon estimate isn't likely to make much of a difference. Keep it simple and stick to the scale.

Vote independently

It's human nature to be influenced by other people. If a technical leader says a story is a 2, it's likely that the rest of the team will follow his lead. For this reason, I prefer an estimation process that lets each team member vote independently. This can be done by writing estimates that no one reveals until everyone is ready.

Another option (that I prefer) is to give your estimation rock-paper-scissors (RPS) style. In our estimation meetings, we talk about a story until we are ready to estimate, then we all "throw" our estimations the same as you would "throw" rock, paper, or scissors. What I mean by "throw our estimation" is that if we think it's a 1 we point one finger. Likewise, a 2 is two fingers and 4 is four fingers. If you need to throw an 8, you can use both hands.

Take the largest estimate

Even when reminded, developers seem to have a hard time estimating with a team in mind. If a developer thinks they can do the story in one day, they throw one finger. Unfortunately, that developer may not be available to do the story, and then some other team member is stuck working on a story that they thought was a 2 or even a 4. I prefer to always take the largest estimate thrown by any team member. You may consider this to be sandbagging, but in reality it's likely that each team member has identified different risks and the team member with the largest estimate has probably correctly identified that there is more risk than the other members have thought of.

Taking the largest estimate has additional benefits. If you must agree on a lower estimate then the team member with the larger estimate will need to discuss why they chose a larger value. This discussion can be uncomfortable for developers who are less senior on the team. They may not know how to do something as quickly due to limited experience with the language or tools. Their skill level often justifies their concerns, and it would be unfortunate if they felt uncomfortable giving their true estimate because they were afraid to discuss why they felt it was higher.

Any discussion of taking a higher or lower value may lead to the entire team raising their value, or it may lead to an inexperienced developer pressured into uncomfortably lowering their estimate. Either way, you'll need to spend more time talking and you wont have gained anything.

Finally, taking the largest estimate can help save time in an estimation meeting. If any member of the team believes the story is an 8, he can speak up at any time while discussing the story and announce that he is going to throw an 8. Unless someone else believes that there is a large estimation gap among team members, there's no reason to continue talking about the story since it will ultimately become an 8 anyway.

In the end, it's consistency that matters. You always know how many stories you expect to get done in an iteration by tracking velocity. Velocity is defined as the number of points you've completed over the life of the project divided by the number of iterations. Your velocity indicates how many value points your team can expect to complete in an iteration. If your estimates are bloated, your velocity will also be bloated. Bloat has no effect on planning as long as the estimates remain consistent through the iterations.

Large estimate gaps

When estimating, the entire team hardly ever agrees on the size of a story. You know by know that I like to handle the mismatch by always taking the larger estimate. However, sometimes a large gap represents a misunderstanding. For this reason, any time there is a two-value gap in estimation, additional conversation always occurs (e.g. a team member throwing a 1 while another throws a 4 requires some clarification). Discussing large gaps also ensures that taking the largest estimate has less chance of being abused.

Insufficient information

On occasion a story may need to leave the meeting without an estimate. It's better to ask for more information than to give an estimate that you are uncomfortable with. An estimate of 8 implies that it's a large story, but you expect it to take twice as long as a 4. Don't simply estimate ill-defined stories as 8's, because you will likely be expected to get it done in the same amount of time as it takes to complete two 4 stories. The goal of an estimation meeting isn't to estimate all the stories, it's to provide estimates on the stories that offer sufficient information for an informed decision.

Required involvement

No one enjoys estimation meetings (okay, no one I know). In my past projects, the fastest reader would read the story aloud, the developers would ask the domain experts questions, and then they would estimate. When the developers weren't quizzing the domain experts, the domain experts usually did other things on their laptops. At first glance, I thought this was a good use of their time, but things got missed. Later, I joined a project whose manager insisted that we go around the room and make everyone read a story when it was their turn. Suddenly, the domain experts were engaged because they were worried about looking silly when it was their turn to read. The meetings became much more valuable due to everyone's involvement.

Pigs and chickens

In a ham-and-eggs restaurant, the pig is committed but the chicken is simply involved.

I often hear that the business shouldn't influence developer estimates because developers are pigs and the business is full of chickens. I think this is a bad analogy. It's more likely that a bad product will get the business team fired than the technology team. I'm sure the business feels just as committed as the developers. However, it remains a conflict of interest to let the business interfere with estimates.

It's as simple as this: the business wants to know what functionality they can get in the next iteration. To know what to expect, they need estimates. Since the business will not be writing the code, they cannot contribute to proper estimates. The more they are involved in the actual estimation, the less likely it is that they will receive realistic estimates. The best domain experts answer questions in meetings but never assert in any way the level of effort it will take to complete any given story.

Group size

Teams come in many different sizes. On smaller teams of six or less, I suggest the entire team attend the estimation session. The many points of view are likely to solidify vision and positively contribute to an estimate. However, I believe there is a point of diminishing returns. Not everyone on a large team needs to take part in estimating every story. Additionally, an estimate produced by six people should be just as accurate as one that comes from 15 people. If your team is larger than six people, I suggest breaking into smaller groups for estimation. In general, I like to get at least three people to estimate any given story, but no more than six.

New stories

New stories come in two forms: new feature requests and stories that split. I generally wait to estimate new stories based on their priority. If a story needs to be done in the next iteration, it generally requires an immediate estimate but if a new story isn't going to be played for several iterations, it can make sense to hold off until you have enough stories to justify an estimation meeting.

I find estimates from estimation meetings to be more reliable, since they come from an environment where everyone is focused solely on estimation.

Stories resulting from a split provide an additional complication: they likely already have an estimate. I strongly suggest that the new stories be estimated without considering any previous estimate. If a story carried so much risk or uncertainty that it required splitting, it's not likely that its estimate was realistic – ignore the original estimate.

No laptops

At least, do not permit developers to have laptops during an estimation meeting. Print the story list for everyone or project it on a screen, but don't ask the developers to read the story list from their laptops. Laptops almost always find ways to distract developers, thus taking away from the goal of the meeting: getting valuable estimates.

Required participation

This suggestion is important. In theory, no developer from outside the team should be attending an estimation session. That means that every developer that attends an estimation session will potentially be tasked with working on a story that's being estimated. If a developer is not comfortable estimating a story, then I'm not comfortable with them working on the story. Of course, there are exceptions. I generally give new team members one week to come up to speed before I ask that they participate in an estimation session. But, in general, a developer who refuses to participate in estimation should be a clue that there's a bigger issue that needs to be resolved.

Stale estimations

Teams change, projects change, and random events occur. Whatever the reason, estimations can get stale. Stale estimations don't help anyone. The development team feels pressure to deliver to outdated estimates and the business expects stories to be completed according to projected velocity. It doesn't matter why estimates get stale, what matters is that the estimates are no longer realistic and the plan is no longer reliable. I've never been part of a project where the estimates didn't go stale within 12-24 weeks. It's better to admit that an estimation is stale than it is to plan with inaccurate information. For this reason, I suggest revisiting any estimate that was reached more than 12 weeks ago. The estimate will hopefully still hold true, but giving the developers an opportunity to speak up given new information is nothing but helpful to the business.

Bribes

This is the easiest suggestion of all: bring high-quality snacks to all estimation meetings. Sugar has been scientifically linked to happiness, and happiness leads to collaboration. It's the simplest and cheapest possible way to make an estimation meeting something to look forward to. Keep in mind though, that high quality is the key. If you bring the same snacks that are already sitting in the team room, it's not very exciting. On my last project, I went to the bakery for fresh-baked cookies every time I remembered there was a meeting.

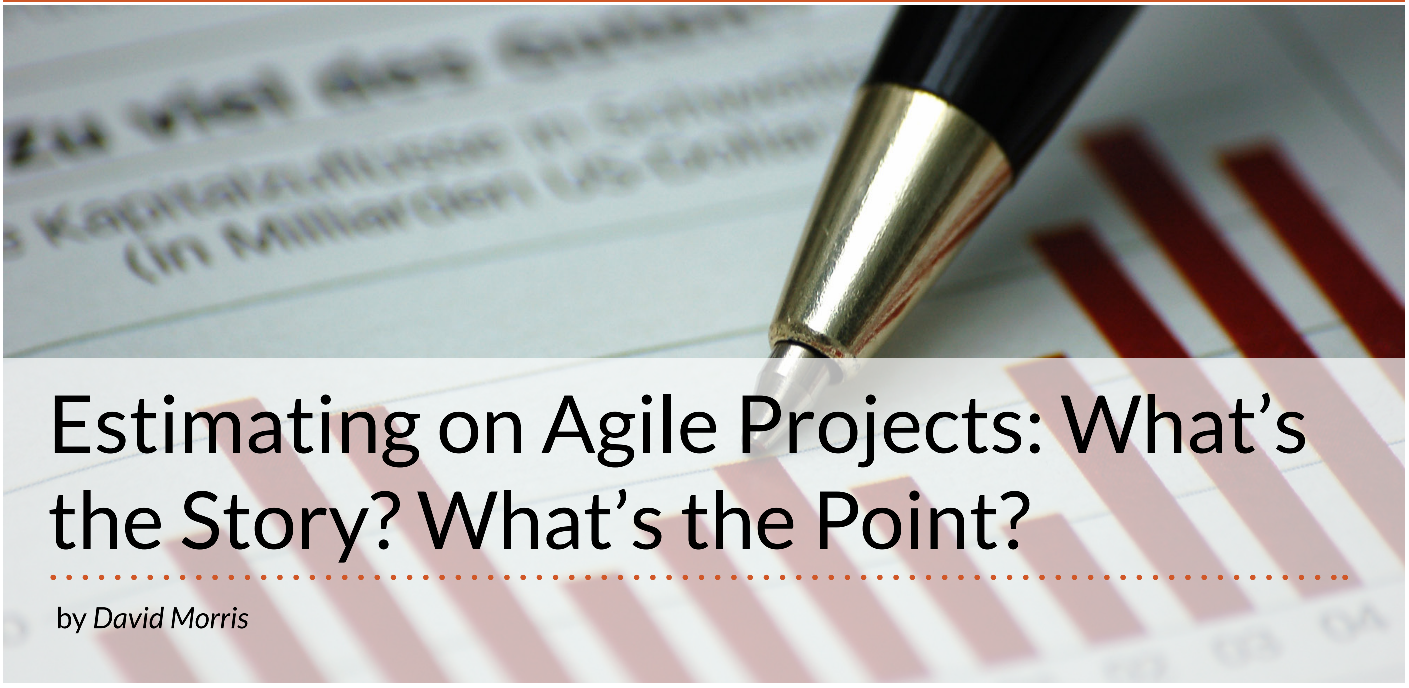
Credit

I'd like to give special thanks to Brent Cryder, Dennis Byrne, Fred George, Joe Zenevitch, Mike Ward, and Sean Doran for helping me evolve and solidify these ideas. Just like every other list of people, mine surely leaves out other contributors. Please forgive me for leaving you off.

ABOUT THE AUTHOR

Jay Fields is a software developer and consultant at ThoughtWorks. He has a passion for discovering and maturing innovative solutions. His most recent work has been in the field of domain-specific language, where he has delivered applications that empowered domain experts to write domain logic. He is also interested in maturing software design through developer-testing and software-testing in general.

READ THIS ARTICLE ONLINE ON InfoQ



Estimating on Agile Projects: What's the Story? What's the Point?

by *David Morris*

Introduction

Before you commission a painter to decorate your home or a mechanic to fix your car, you get an estimate from them, right? You need to know how much it's likely to cost and how long it might take. It's just common sense.

What does experience tell us, however? How close are those original estimates to the final bill? It's all too likely that the painter will find loose plaster that needs removing and that the wall needs rendering and re-plastering. The mechanic is sure to find additional work required to get your car roadworthy again. In a 1951 cartoon for the New Yorker magazine, Syd Hoff drew a mechanic saying to his customer, "Of course that's only an estimate; the actual cost will be more."

If the painter or mechanic tells us soon enough, we can choose not to take on the extra work… and yet, all too often, we feel we have to fix these additional things. Who wants to live in a house with a potentially damp wall or drive in a car with potentially faulty steering?

How do we overcome this? A common reaction is to insist on a full and final fixed-price estimate, or "quote" as it's commonly called, so the tradespeople work harder and longer to get more accurate estimates. Yet however hard they work, they still cannot really predict the unexpected.

In the life of projects, this is just the same

Traditionally, project managers tend to focus on creating detailed estimates that can withstand scrutiny from the finance team. Of course, this is based on "known knowns" with some contingency for the "known unknowns" – and as Donald Rumsfeld is famously quoted, "There are also unknown unknowns – there are things we do not know we don't know." Like the tradespeople above, we can never really predict the unexpected.

The more we invest in creating elaborate estimates, however, the more problems we cause. Detailed estimates can be seen as a binding quote, a target that distracts from delivering value, and focusing on delivering something – or even anything – to the agreed date and cost.

We beat ourselves up in post-implementation reviews and tell ourselves to just try harder; Einstein, however, defined insanity as "doing the same thing over and over and expecting different results" so there must be a better way.

Surely this isn't true for agile projects? Don't we just start with a high-level scope and work it out as we go along? Well, yes and no (or as they would say here in New Zealand, "Yeah, nah"). While we don't hamstring ourselves by creating detailed estimates, it is still vital that we get a feel for the size of the work we're considering, and here's why…

Why we need estimates

Forget about estimates being used to build beautifully crafted Gantt charts that force us to focus on tasks and not outcomes. There are three outcomes that need us to get a rough handle on how big something is.

When we're considering the justification for a proposed project, we need to understand the likely costs in advance, so that we can decide whether it is worth the investment.

When we're launching new or improved products to market, we need some idea of roughly when significant features might be ready for release, so we can plan associated activities.

When we're prioritising work, the product owner needs to understand the cost, and the team needs to understand the value, of each story (or backlog item).

It's also interesting to note that estimating can be a really healthy activity, so long as the whole team collaborates on it together. It helps foster a buy-in from the whole team and ensure that everyone gets a common understanding of the scope and value to be delivered.

However, the label of "estimates" can be distracting.

Use sizes rather than estimates

To avoid setting an expectation that we're talking about cost and time, when we estimate the complexity of a story, some of us like to refer to this as "sizing" rather than estimating. In the '90s, when I first used scrum and XP – before they were even called agile practices – we sized stories using T-shirt sizes (S, M, L, and XL).

Now, however, we use story points – a way of sizing stories relative to each other – so we find what will be the simplest story, size that as 1-point story, then compare another story to that, if it's more complex is maybe a 3-point story.

To make things more interesting, we don't use a simple sequence like 1, 2, 3, 4, 5, etc. Instead, we use a modified form of a Fibonacci sequence like 1, 2, 3, 5, 8, 13, etc. (as seen in The Da Vinci Code). This makes the jump between numbers larger the higher we go and drives us to choose which is smaller or larger.

While this is not an exact science, it is more than good enough for the three outcomes mentioned above, and as John Maynard Keynes said, "It is better to be roughly right, than precisely wrong." This means, though, that we still need to translate story points to rough time and rough cost.

Another practice central to agile projects is establishing the definition of done, that is to have a comprehensive understanding of everything required to say that a story is done and releasable, including items like user documentation, translation, advertising, etc.

Provided we have a good definition of done, we can then take a couple of sample stories and calculate the effort required. From that we can get rough cost estimates for an investment decision, rough timing for release planning, and enough understanding to assist in story prioritisation.

Some, however, still find estimating by story points a distraction, and one response has been the debate around #NoEstimates.

So what's with the #NoEstimates debate?

If you have been following any of the major influencers on Twitter, you might have noticed some getting involved in discussions with the hashtag of #NoEstimates. While this sounds like a call to cease estimating altogether, it is really a call to stop sizing stories and instead just start developing.

This discussion arose, and has gained momentum, because the experience of those working on large projects has been that whether you size by story points or just count the number of stories, the tracking of throughput is about the same.

While this might partly be down to the more experienced delivery teams decomposing stories to consistently smaller sizes on a just-in-time basis – as this enables greater throughput of potentially shippable increments – the reporting looks similar even on projects still working with a range of very small to moderately large stories in each iteration.

While this makes story prioritisation simpler and enables the delivery team to get on with it quicker, our poor product owners and sponsors still need to know roughly how much it will cost and how long it will take. The good news is that we can still do this based on counting the number of stories, once we have sufficient metrics to make this feasible.

For any new team starting out on your agile journey, however, I would still strongly recommend you size with story points until you've reached the experience and maturity levels of these teams.

Conclusions

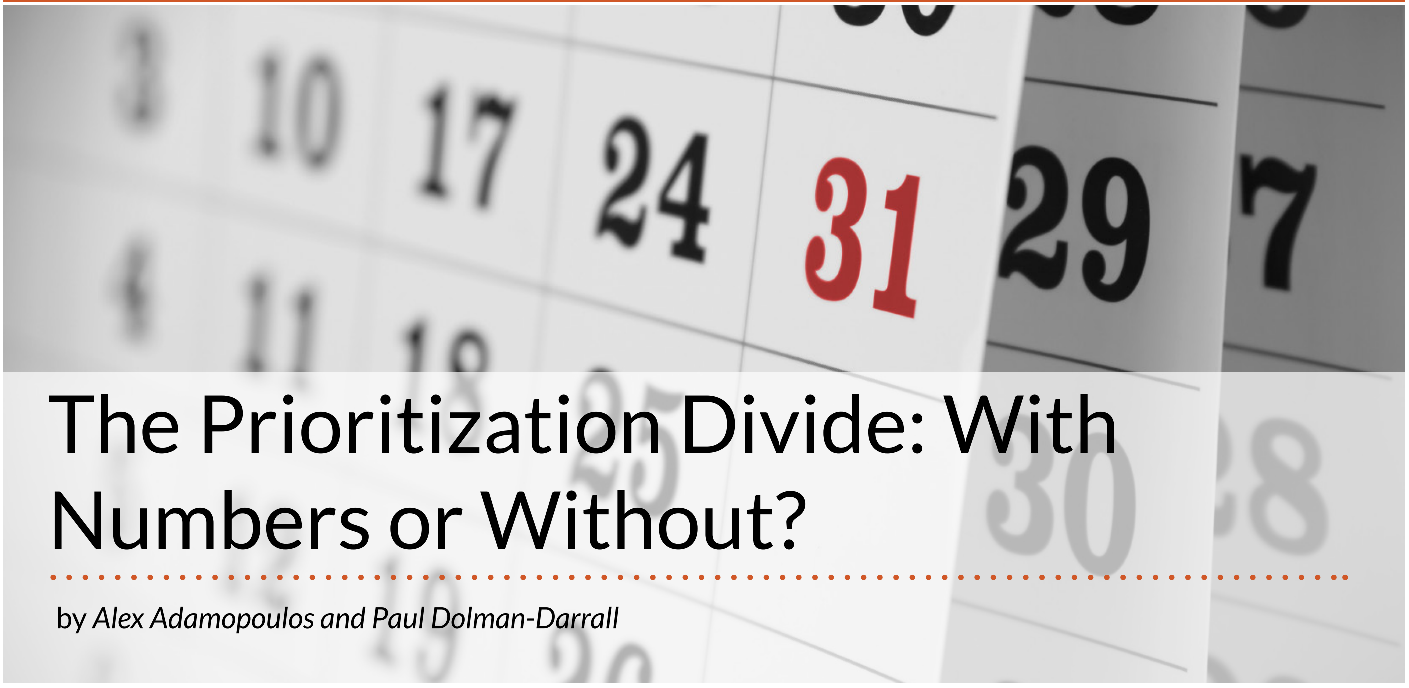
So to sum up and to mangle a famous quote from General Dwight D. Eisenhower: "In preparing for [projects] I have always found that [estimates] are useless, but [estimating] is indispensable." We still need to do it, whatever we call it and whatever we're counting when we do it.

I am indebted to Esther Derby (estimates become targets), Mike Cohn (estimates for release planning and prioritisation), Ahmed Sidky (whole-team estimation), Martin Fowler (story points), Ian Mitchell (definition of done), Vasco Duarte (story count vs. story points), Stephen Forte (metrics vs. estimates), Neil Killick (experienced teams define smaller stories), and many others for sharing their thinking on this topic – and I have linked to their respective articles in context above.

ABOUT THE AUTHOR

David Morris is an independent business agility practitioner, coach, and instructor, working through his company Sophorum, delivering business analysis, team leading, coaching, and training services to customers throughout New Zealand. With nearly 30 years' experience in project delivery, he has worked in and led teams and run his own business across strategic, business, and technical projects following structured, iterative, and agile methodologies.
David is a qualified CBAP, ICAgile Certified Practitioner, Certified ScrumMaster, and an IT Certified Professional. He runs study groups, professional development workshops, and boot camps; helps organise events for Agile Auckland and IIBA NZ; has spoken at conferences and events in Europe and Australasia; contributed to several books (including Agile Extension to the BA Body of Knowledge); and is an active blogger, tweeter, and Wikipedia editor.

READ THIS ARTICLE ONLINE ON InfoQ



The Prioritization Divide: With Numbers or Without?

by *Alex Adamopoulos and Paul Dolman-Darrall*

How do you prioritize when you start development? Do you assign each story a dollar value based on expected revenue or savings? This may work when you're discussing a major feature, but when you drill down to moving a dialogue box to aid user experience, how do you determine its value?

Or do you select stories according to "Must, Should, Could, Won't" and then assign them a relative value with story points or labels? It's not difficult to do, but does it really select the most critical or highest value out of all the stories that are crammed into "Must"?

There are many methods, but a basic divide runs through the heart of prioritization: do you do it with numbers or without? There are arguments for and against both positions, but instead of examining these, people tend to fall naturally into one camp or the other. Once there, they can become quickly entrenched in the belief that the other camp is foolishly mistaken.

Those who criticise the numbers approach say "Those number-crunchers spend so much time finessing their estimates that they don't get any actual work done. By the time they've calculated a cost of delay, they're delayed already."

Those who prefer numbers mock the alternative: "High value/low effort? What kind of subjective, gut-feel way is that to run a business? Why not just throw the cards into the air and develop them in the order you pick them up?"

Consider:

What's your natural preference when prioritising? How about for your team and colleagues?

What prioritisation methods have you used on different projects? List the main advantages and disadvantages as they appeared to you.

The most common model today

Although there are numerous prioritization models in play, one of the most commonly mentioned on discussion boards and in interviews involves assigning relative points. Essentially, the team gets together and assess the value of a whole bunch of stories. The lowest-value story on the table becomes the baseline, and other stories receive points relative to that. Next, the technical guys give relative effort points to the stories: the easiest story on the table gets 1 point. Is the next story three times as hard? Give it 3 points. Some teams use T-shirt sizes, Fibonacci numbers, and planning poker, but these are just frills.

The method makes it easy to create a relative priority. Take the values, divide them by effort, and voila – you have your order. As the team begins to work, they establish a velocity. Meeting overheads go down because team members can simply pick from the pre-agreed list according to their velocity. Every few weeks, the team can re-evaluate whether the order is correct or if values and estimates have changed based on the team's performance.

Doesn't this sound like the perfect combination? You get a comparative figure but it's nice and simple so everyone can grasp and enjoy using it. Great!

But there are a couple of basic problems with relative figures, both of which stem from the fact that they are not based on real numbers.

My value of 10 points divided by 2 effort points gives me 5. This is clearly better than a value of 3 points divided by 3 points, which gives me 1. But nothing in this relative term tells me that my values mean anything. If a value of 10 equates to only $20, while 2 effort points equates to two days of a developer's time then the project is not going to make me any money. In fact, I will go bust very fast.

Relative value hides a whole host of assumptions, which are never laid out for us to examine. Is this figure based on revenue? Does it take into account urgency? Does it ignore risk? Because the label hides the work that should go into deciding whether something is valuable or not, it turns out to be just as subjective a measure as saying "It's all important," or "The customer is going to love this, I just know he will."

Observe:

If you have used this method, go back to a previous project and find the original value and effort labels. Now look for the real figures. On a very granular level this can be difficult, but at a feature or epic level, there are probably user figures and thus dollar values assigned against a part of a product. Compare these and look for any large variations – a feature considered valuable but which turned out not to be used, for example.

What assumptions lay behind the original value assignation? Would it have been different if these had been explicit or you had a real cost figure at the beginning?

What lessons can you draw from the comparison? What actions can you take to improve accuracy in the future? Don't forget actions you can take now – if a feature is not being used, delete it!

The value of numbers

'When benefits are not quantified at all, assume there aren't any," advised Tom DeMarco and Timothy Lister in their 2003 book, Waltzing with Bears: Managing Risk on Software Projects.

That's the numbers attitude in a sentence. If you're inputting a feature then it should have a proper justification, and this justification needs to take account of the different elements that make up a label like "value". It should take account of future and present revenue, savings, urgency, risk, and learning. How you express this figure, whether as a dollar quantity, cost of delay, or cost/benefit ratio, is up to you.

Sometimes the calculation is simple. Our new system will automate data entry, so we will save the salaries of the five data-entry clerks currently doing it manually. Their salaries form the dollar value of our system to the company.

You can usually make a good estimate even if you have to make a few assumptions. Good design is often referred to as "intangible", but improvement to user experience should be measurable. Do people get to the registration page and then fail to complete registration? Do we think that redesigning this page will improve registration by 50%? How much is each registration worth on average to the company? With these questions we can assign a dollar value to a redesign even if the change is quite granular (moving a dialogue box or changing filters).

Look in these places to try and find what type of value your product is delivering:

Increasing revenue

Savings (These might be direct or equivalent, i.e. if we don't have this feature it will cost us $X to do it another way.)

Protecting revenue (i.e. Without this feature, we can expect revenue to drop.)

Protecting against costs (i.e. Without this feature, we will be exposed to certain costs.)

Why use numbers?

Numbers make the argument more objective. Rather than arguing about whose idea is better or more important, teams can use numbers to change the conversation. It becomes clear that ideas are competing against one another based on their value to the organisation.

Numbers can speed decision-making. Once you have an economic framework, you can express many trade-offs as decision rules. For example, you might decide that if the cost of delay is more than two times greater than the cost of the resource required to avoid that delay, the team should be empowered to incur that cost. That might mean hiring more staff or investing in automation. Power has been devolved to the team but managers retain overall control because they set the framework within which decisions are made.

Numbers don't need to be difficult. There's no need to strive for a spurious exactitude. The point is to avoid big errors, not to sweat a decision between a project with a $3,000 cost of delay and one with a $3,100 cost of delay.

In general, where you can come up with a number, it is worth trying to do so.

Act:

For a current project, pick the top three items and try to work out a real value for them. Don't forget to consider four factors that make up value: financial (revenue and savings); cost; learning; and risk.

Check the figures with business owners then file them away. You will need to compare them with reality once the product has launched.

If you can come up with the worth of the feature or project to the business, then you can also calculate a cost of delay. Are there any decision rules that this might help you make? For example, might this justify overtime or investment in automation? Try to get the decision rule approved in advance so that you can act swiftly if anything occurs to delay the product.

The problem with numbers

It seems simple: use numbers more often. But it's not quite that straightforward because there are a few, very real disadvantages to using numbers of which you should be wary.

Numbers feel like unassailable ground. That spreadsheet of future earnings looks so convincing and took you so long to set up (plus it has these nifty little macros).... People often fall in love with their projections – so much so that they are reluctant to take new information on board. Instead, they cling to the spreadsheet like a drowning man and refuse to adapt.

It's easy to game the system. Most development teams know what a project requires to win approval at a phase gate. It does not take much to just tweak the assumptions (increase conversion by 0.5%, reduce costs by 5%) in order to help the project over the hurdle.

All numbers are based on assumptions. Your figures are only as good as your estimates and guesses – and there may be serious flaws in your assumptions.

So what are the answers?

Transparency and review. You have to record your assumptions and widely share them. You thought that your system would save the salaries of 10 people, but the HR manager has pointed out some redundancy costs that you need to take into account.... You need to review your assumptions as you go, inputting real data as it arrives.

Test early, test often. The only way to gather real data is to get feedback on your model. This doesn't mean just pushing out a prototype and seeing if people like it. Instead, you need to test the assumptions on which your business model is built. Many companies at the moment assume they need to be on Facebook. They rarely ask what it will do for them. Increase customer involvement? Reach a new audience? Even more rarely do they try to quantify these to see if the cost of having a staff member permanently responding to Facebook posts is justified.

Should you always use numbers?

There are only a couple of situations in which numbers are not the most useful technique. Unfortunately, these occasions tend to be common in software development.

Innovation

When you're launching a well-understood product into a familiar, established market, you can have fairly firm assumptions. If you're about to set up a deli in Manhattan, then your model is going to be pretty much like every other sandwich shop's. That doesn't mean you'll succeed (maybe you don't make tasty sandwiches), but it does mean you have a fairly good idea of operating margin, daily sales, and likely costs.

In an entirely new market, your assumptions are so uncertain as to be almost valueless. When Eric Ries set up IMVU, the team had no idea how many people would want a 3-D avatar. Customers themselves didn't know, which meant that focus groups and surveys were utterly useless.

At such a point, it makes more sense to record your hypotheses and then test them one by one, using numbers or not. These need to include a business element as well as a technical one. Ries, for example, had assumed that users would not want to bother setting up new contacts and that therefore the software needed to integrate with existing platforms. This turned out to be wrong. Testing that assumption was more important than the fact that sales were not as healthy as Ries had hoped, but it was the low numbers that had alerted the team to the problem.

The team doesn't own the numbers

Many commentators talk about the process of estimation as waste: the more time that estimation takes, the bigger the waste. There are some circumstances in which this is true – and unfortunately those circumstances are not uncommon in IT.

Picture this: a project-management team spends three months creating a list of requirements and then assigns a numerical value to each. They hand the list over to the development team and ask how long the project would take. After due consideration, the development team announces that they think a good estimate would be two years. The project manager flings her hands up in horror. "That's way too long! We need it in six months!"

In this example, the estimation process is a waste. The team needed to establish the true constraint up front: the six-month timeframe. This is not very unusual, but it permits the team to usefully establish the the fixed cost of delay and make decisions based on that, whether they choose to reduce scope or increase capacity.

Apply learning:

For a new project, think hard about which prioritisation approach would be the most effective. If you decide to go without numbers, ensure that you have a suite of tests that will provide early feedback on your assumptions. Write those assumptions down and share then with the team. Keep them visible as the work progresses and change them as you go. If there are no changes, this is a sign that you could have used numbers up front because you had firm assumptions. Start assigning dollar or cost of delay values and check your prioritisation – it's a good, rigorous discipline.

If you choose to use numbers then make your assumptions explicit and have them visible for review. Keep testing your figures and your assumptions. A willingness-to-buy study can be as simple as emailing all your contacts to describe the product with a request to reply if they're interested. This is free to do, and although it's almost certainly an overestimation of interest (these are warm contacts and they're not being asked for credit-card details), it's better than a guess.

Conclusion: Love the figures of failure

Numbers help – except when they don't. Oh, what a helpful statement to take away from this! Let's try to boil it down to something you can use as a rule of thumb: most teams should be translating their value labels into real dollar figures more often than they do. Why?

An estimated cost of delay, a revenue projection, or a predicted cost saving may not be accurate. Indeed, you might get them wildly wrong. But their clarity is designed to help you focus your efforts on a visible, explicit, and objective set of assumptions, which you then test through early and frequent feedback.

If you believed that 10% people would buy your product but a test reveals that only one in a hundred agree to try it, you know that there is something very wrong. You might conclude that the product is a bad idea and kill it. You might learn something important and pivot to take the product in a new direction. Or you might decide that 99% of people just didn't understand your genius, so you will continue anyway and raise the risk.

The numbers don't tell you what to do. Their purpose is to provide an objective check on your assumptions. If you receive 10 emails saying people love your product, it's easy to feel that everything is going well. Only when you compare that number (10 out of 1,000) to your sales projections can you place the good feedback in context.

Numbers help transparency, feedback, and objective decision-making.

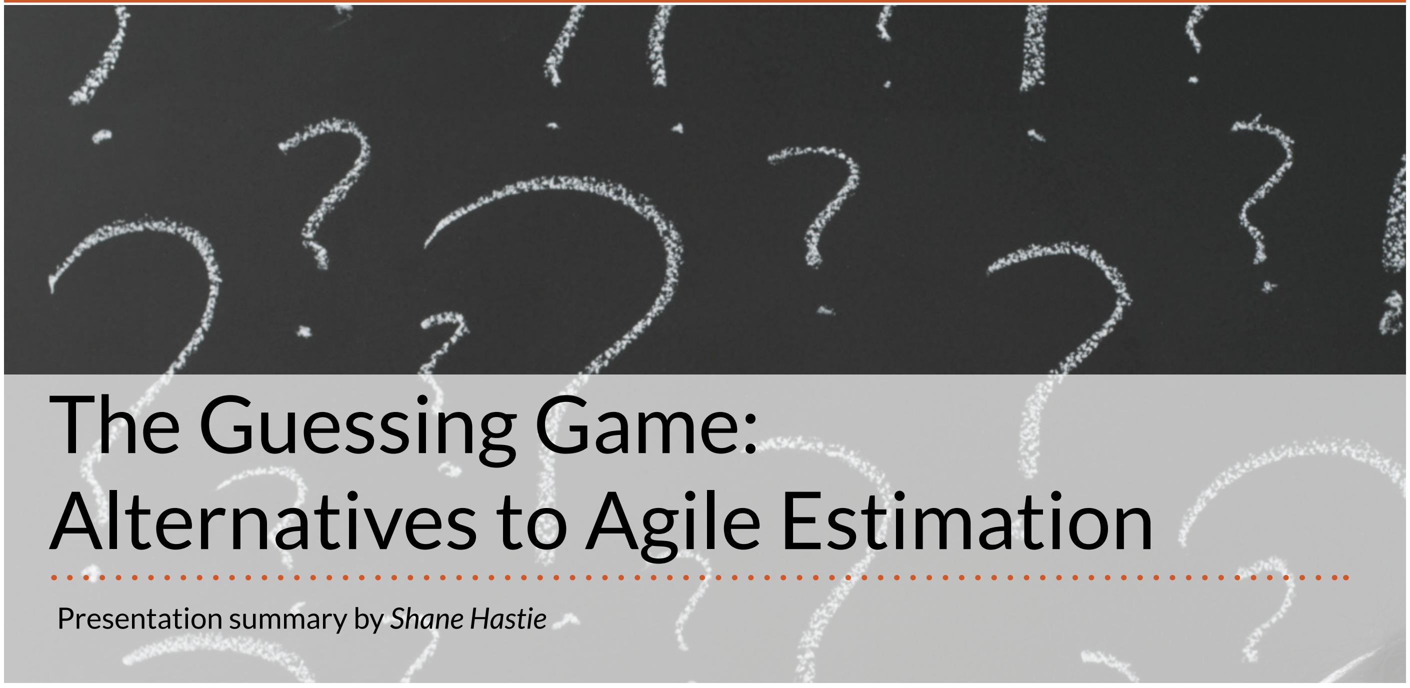
That's why they might just help you avoid the painful sight of a roomful of managers looking around for someone to blame when the game-changing project sinks like the lead balloon you always feared it might be.

ABOUT THE AUTHORS

Paul Dolman-Darrall is an IT director known for developing people and successfully leading large global teams across various change programs for some of the largest companies in the world. He has contributed to strategy of government. At Emergn, in his role of executive vice-president, he has helped launch Value, Flow, Quality (VFQ) education, a work-based learning program to help practitioners achieve immediate business results through the application of skills in practice. The program is designed to help IT departments and business leaders who rely on technology to put in place smarter, more effective work practices to facilitate change, generate significant return on investment, and inspire innovation in practice.

Alex Adamopoulos is an executive with more than 25 years' experience in global services organizations. He has extensive international experience with a deep understanding of culture, work, and life ethics especially in relation to establishing alignment and crossing cultural barriers. Over the years, Alex has brought know-how and practical business experience to companies that want to excel and compete globally. With a focus on performance measurement, business value, and bottom-line profitability, Alex has successfully applied working models and practices to accelerate the solutions and strategies of companies to drive results.

READ THIS ARTICLE ONLINE ON InfoQ



The Guessing Game: Alternatives to Agile Estimation

Presentation summary by *Shane Hastie*

At the Agile Australia conference in 2013, Neil Killick presented a talk in which he proposes ways to reduce risk and uncertainty, calculate a product's price, determine delivery dates and roadmap, and do scrum and XP without using estimates.

He set the scene with the need for some form of predictability about software development.

> *With estimation, in software, which is what essentially I am talking about today, the question we typically try to answer is what am I going to get and when. That is a perfectly legitimate question. If we are investing some money in something, we all want to know what we are going to get and have an idea of when we are going to get it. So, this talk is not about saying we do not need to answer this question - we absolutely do - but it is about seeing that there are other ways of answering this question, maybe flipping the question around, and using alternatives to actually fulfilling our expectations when it comes to building software.*

He went on to discuss the difference between estimation and guessing.

> *The first thing I want to mention is, I guess, the distinct difference between estimating and guessing. When we make a guess, we are essentially using gut feel. However, we do not have any knowledge or empirical data about the guess we are making. Whereas with an estimate, we are using some kind of knowledge, whether it's tested knowledge or real data, to make a prediction. I want to really question when we are estimating software-development projects or products: are we making an estimate or are we actually making a guess?*
>
> *It is a very important distinction and it is something we need to think about when we are putting together business cases and having to estimate development cost, or if we are a team that has been assigned the projects. Are we making guesses or are we actually using some kind of knowledge to base that on?*

He mentioned the dangers inherent in estimating, managing expectations, and treating estimates as commitments.

> *Whether we like it or not, if we estimate a project up front, we are setting an expectation level. Everyone has an expectation about what they are going to get at the end and this may vary between stakeholders and customers. It is a dangerous game to play because we want to make sure we make our customers and stakeholders happy. If we are using their expectations before we start, at the beginning of the projects, how do we know we are going to deliver on their expectations at the end?*

So, we have to be very careful to make sure that we are aligning what we are doing with the expectation level. This is really something we need to be looking at as we move along rather than basing it on what we do up front.

Of course, if we are using an estimate as just an estimate and saying, "Well, we know this is going to change because it is an estimate," that is fine, but I do not see that application of estimates in software development. I actually see a dysfunctional model in which we are setting deadlines and promises based on these estimates. So, these expectation levels actually get set and make it really, really difficult to change as we go along because we are now sort of afraid we are going to alter the expectations of our customers in a way they are not going to be happy with.

He explained how estimates become self-fulfilling prophecies.

Estimates are self-fulfilling prophecies. What I mean by this is that say we estimate a software project will take 12 months and six months in, we realize we have made a big mistake and that it is going to take us a lot longer, maybe even two years. What that will do, given the culture of setting promises and deadlines based on the estimate, is affect behavior. We are going to modify our behavior based on the estimate, rather than on trying to build the right thing for our customer and deliver, to exceed the expectations of our customers.

So, it is self-fulfilling because we will say, "Okay, we are six months in. We are not going to hit 12 months. What are we going to do?" One thing we could do is cut scope, or another is that we could start cutting corners with our work. With this kind of pressure, the closer we get to the deadlines, which are these kind of arbitrary deadlines we have imposed on ourselves, the more stressed we are going to get and the more we are going to cut corners and quality is going to suffer.

On the flip side, we could be six months in and we go, "We are going to get this knocked off in a month. We said this was going to take 12 months, so we have a year's budget allocated and we actually don't need that." So now we change our behavior in another way. We start saying, "Well, we still have four months' worth of money. Let's build some more stuff. Let's start gold-plating what we have built and build more features." Essentially, there is a danger that we are building stuff that is not going to be used and that the customer does not want. We are just building those things because we have set the expectation that it was going to take a year and this is what you are going to get in a year.

So now, we are altering our behavior and in neither those scenarios is the 12-month estimate going to come true because we have altered our behavior. The project parameters have changed. We know that. When we are building software, it is variable – it is a creative pursuit. We know that the premise is going to change, but what is also going to change is our behavior based on our progress towards that target.

Killick described a typical dysfunction he has seen when teams take estimates as commitments.

Essentially, we come up with a good idea that we think is going to be valuable for the business and we present a business case. We then have to come up with a cost and a value to this so that we can sell it to the executives. So, we need to calculate our return on the investment and show that it is something that we should be doing. Now, at this point, we have to come up with how big a team we are going to need, what are we actually going to be building, and what that team makeup is going to look like.

I really question at this point how much knowledge we actually have to create an estimate. Are we, in fact, making a complete guess and using our biases to try to get the business case over the line and it does not matter because once it gets approved it's someone else's problem? It will get allocated to a team, maybe months, even years later. I have seen this process take two years going from a business case to a team starting work.

So, the business case gets approved and prioritized and eventually a team gets put together with a budget to work within. Even though that decision has already been made by the business, the team now has to estimate what they are going to build and how long it is going to take.

What happens if the estimate is too big? So, the team has a year's worth of budget, but comes back to say, "Look, this is a two-year project." Again, there are things we can do here. We can say, "Let's reduce the scope, see what the real must-haves are and get rid *of some of the nice-to-haves." We might be able to reduce scope enough that the team answers, "Yes, that is all good. That will fit in a year," and the project gets approved. If that does not happen and we end up in a situation where the estimate is way too big and we can't do anything about that, then we can potentially ditch the whole thing.*

This does not happen often, but I do see this happen and both cases are quite dangerous. If we approve a project based on a year's worth of work, and we estimate it and we think it is all good, there is a lot of risk inherent in there. But also, and this is worse, we might ditch doing something really, really valuable for the company because we have clumped together all this value into a year-long project and we are making a really big decision – to go or no go – based only on that clump.

I was asked once to run an estimation session where this exact thing happened, where we went away, estimated a bunch of requirements, and came back and said, "Look, this is a year-long project," and the whole thing was canned because of that. I heard "It is far too long, we are not going to be able to do this. This is going to cost too much." After the session, I asked a couple of the product owners in the room, "Would any of those requirements, if we just built one of them now, provide good value to you?"

"Yes," they said. "This thing we have prioritized as number one. If we were to do this, it would be immensely valuable. It would save us loads of time and make our lives a whole lot easier." That thing we had estimated to take between one and two months' worth of work but because we treated the project as one big, fat clump of value, we ignored the whole thing even though we could have generated lots of value by just doing one or two things from that list.

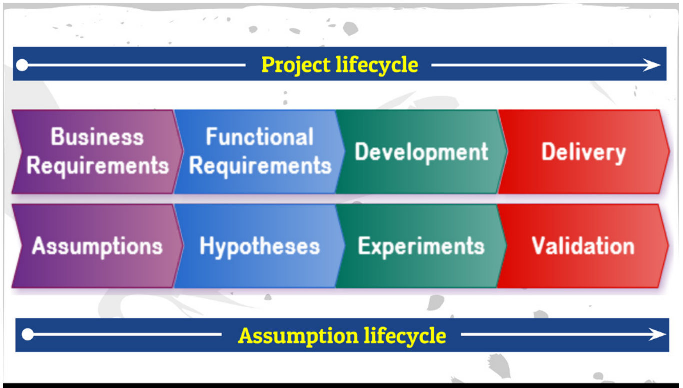
He suggested that there is a need to change how we look at value.

We need to start thinking about what we can do now. What is the most valuable thing we can do? We need to be really serious about that rather than putting all those things together and making big decisions based on the whole.

So, we have done the conventional thing and we have prioritized our project and because we are in uncertain world, we limited our options to two or three things. We conclude that project A wins because it has a higher ROI.

The thing is, that is all very well, but how do we know that we are building the right thing here? We have kind of confined ourselves to project A versus project B, but there is a whole bunch of other options that we have not considered. Why is it that we can do only project A or project B, or even more pertinently, why can't we do both? Why do we have to look at these as big, monolithic projects? Why can't we look at smaller pieces of value that we can do concurrently?

He introduced the assumption life cycle, which is a way of looking at projects, comparing it to the project life cycle.



PROJECTS MAKE ASSUMPTIONS

Big, up-front project thinking is essentially a lot of assumptions. According to the traditional assumption life cycle, we have business requirements, functional requirements, and they turn into development and then delivery. Actually, these business requirements that we come up with are all assumptions. We do not actually know these things are valid at all. We say, "Well, these are our business requirements. Let's decide how we will deliver those and get some functional requirements around those". These are just hypotheses. Again, we have not yet validated that what we are doing is a good idea. Then, in the development phase, is when we experiment. You are going to see a lot of similarities with sort of lean startup ideas, but at a broader scale, across larger projects. Then, finally, we validate these experiments. Are they valid or not? Are the things that seem to be true actually true or not?

Killick then looked at the impact of duration of the assumption cycle, and how long projects are inherently high-risk activities.

> *If we do this thing in short cycles, say, of a month, there is not much problem with investing too much cost. But, if we are looking at projects that take years or even six months, that is a lot of money and a lot of time to be investing and in that time, so many things are going to change: the market; the way we do things; requirements are going to merge; and things are going to take longer than we thought to build. All of these factors are going to happen and we have not validated our assumptions. Of course, when we get to the end of the project and deliver on time, we celebrate the success of that then realize that we built something that no one wants and that is actually not going to deliver any value for us.*
>
> *So, essentially, we are making really big bets and we are putting a lot of eggs in one basket, and this is an extremely risky approach. We don't do that with our cash in other situations. If we are investing in property or the stock market, we make small bets. We diversify our risk.*

He then tackled the different aspects of project risk:

> *The other thing that is going on here is that when we start a project, we identify what the risks are – I call these the "known unknowns". So, essentially, we know there are things that might go wrong in the projects and we call those things out and those are risks. But then there is also a bunch of things called "unknown unknowns", which are things that will just happen, and we can't possibly predict them or cater for them. And actually – let's call this "emergent value" – this emergent value can be positive or negative. So, as we go along, things are going to happen in the market or even in our own project that are actually going to make us go, "Uh, this is going to cost us more than we thought," or "Actually, this has so much more potential value than we initially thought." These things happen as we go along. We cannot predict these things at the beginning, but we tend to ignore them. We only call out the risks and the sort of issues we know about. We don't call out the fact that there is going to be emergent value.*

Another aspect is how large projects result in missed opportunities.

> *If we commit to one project and we sort of dismiss project B and any other manner of things, we could actually be missing out in opportunities. We could get invested into this project and, even three or four weeks in, some really cool opportunity comes along, but because we have put all our eggs in one basket and basically committed all these people and all this time to this one big thing, it makes it really difficult to be agile and to respond and be pro-active in the market. We are kind of going, "Yeah. This is what we need to do as a business and we are committing to this for six months, a year, two years."*

Having identified the problems and risks associated with the current estimation approach and typical project duration, Killick presented alternate ways to approach the topic by using real constraints.

> *Is there a better way of doing things? The way I like to see this is that when we are estimating software projects, what we are essentially doing is putting arbitrary boundaries around what we are doing – arbitrary constraints. Now, arbitrary constraints are not good for the reasons that I have discussed. They create dysfunctional behavior. What is good in software are real constraints – and not just in software but in other walks of life as well.*
>
> *When we have a real constraint to work with, it forces us to be creative. Imagine if you normally spend $100 a week feeding your family and, for some reason, you only have $20. You are not going to say, "Well, I don't feed my family then." You are going to say, "I need to come up with a way of feeding my family for $20." We start getting creative about this and coming up with ideas and better ways of doing things.*
>
> *So, rather than saying, "What am I going to get and when?" and trying to narrow this thing down and estimate it, let's look at it from another angle and say, "This is my budget. This is how much I want to spend. Can we do this for this amount of money? Can I get the software equivalent of a Ferrari for $500 or $1,000?" Having a budget creates a real constraint. It is something that we can work with. We have to get creative about it because we have no other option.*
>
> *I love the scene in Apollo 13 where the guys have to build an air filter with bits and pieces from the ship and they have to find a way of making this fit into this with only this. Making a square peg fit in a*

round hole. That constraint brought out an amazing creativity in a short period of time.

If we can use real constraints in building software, then we can bring out the creativity in the people we have. We hire excellent engineers and designers and UX people. All of these people are really skilled and it is their craft, their profession. Then we kind of give them the requirements, we give them the solution, and we tell them what to do and put them in a box. We don't give them that freedom of creativity to actually come up with the right answers. So real constraints are really good.

He then explained how iterations are useful when working within constraints.

Once we have that real constraint, we can forget about the end goal for now. Let's start building and learn what we can do without money. Now, if I have $500,000 to spend, the last thing I am going to do is to try and think up $500,000 worth of stuff to do. What I will do is to think about my problem, what I am trying to solve, and then create mini-constraints to enable me to experiment and learn and then see what is actually possible.

So, let us see what we can build for $50k. Let's go away for a month, rather than thinking about the 12-month picture. Let's go away for a month and not just increment over a product backlog. I am not talking here about coming up with a product backlog and doing 1/12th of it. I am talking about coming up with an actual solution for what we are building or for what we are going to do. So, I want to solve my problem in a month.

Now, that does not mean I am going to solve it in as high-quality a way as I would have if I took 12 months, but I might end up with a Holden1 of software after a month and I might look at that and go, "Actually, thinking about it, this Holden does everything I need it to do. It fulfills the need I had and the thought I had and I am going away to drive my Holden and have a lot of fun with it."

Or I might go, "I still want to head towards the Ferrari route and what I can do now is iterate on quality." So, I can now say "Let's go away and spend another month and take this idea to the next level, where we might chop away at the Holden. We might say that was a terrible idea, but we have learned a lot from doing that. So, let's start again."

So, we are properly iterating, we are holistically looking at our problem and solving it in iterations, not just incrementing a product backlog.

He discussed the difference between this approach and the everything-up-front commitment approach commonly used in organizations.

So, we are essentially drip-funding. Compare this to the business case of earlier. In this case, we are presenting a business case; we are approving it as viable options. We are saying "This has potential value," and we think it has more potential value than other things, so we want to go ahead with it. We prioritize that initiative and then we assign a team, and if we have fixed teams that are ready to go and take on this work, we are going to get a better result out of that.

But, essentially, we go away and we do these two-week or four-week – whatever they are – experiments, iterations. We then ask, "Is this initiative valuable enough?" If it is, we continue funding it or we might even go, "You know what? This is bigger than we actually thought this was going to be, guys. Let's scale this up. Let's hire more teams and put more into this because this is potentially more valuable than we thought it was going to be." Or we may say, "This is a dud, not as valuable as something else that has come up, so we are going to pitch this and we are going to switch you guys onto this new initiative."

He explained that the way many supposedly agile organizations work is not iterative but just incrementing over a product backlog.

This is putting the "iterate" into iterations. It's really the key message to get across. When I see agile teams, they are working with product backlogs and many, many of them are hamstrung by the product backlog that was basically derived up front. They are essentially just incrementing through these and, in some cases, even have divided the product backlog into iterations, so they know what they are going to be doing four or five iterations down the track.

1 Holden is a make of car popular in Australia: http://www.holden.com.au/

Well, that is not iterating. That is just incrementing and we are not looking holistically at building up quality.

We need to get frequent delivery and feedback loops. We need to be able to enable that kind of behavior so that we can keep on course, adjusting and making sure we are doing the right thing.

By breaking work down into small chunks we are able to reduce risk and allow companies to respond to emerging opportunities.

So, small bets, experiments, small iterations of things, diversify our risks as I mentioned earlier. They keep our options open and actually let us cover multiple options at the same time, which is really powerful. We no longer have to say, "Oh, these two things look great, but we can only do one of them." Well, no. We can do both. We can do small experiments on both and see which one takes off – or maybe both of them will take off.

He emphasized that good people and stable teams are key to effectiveness.

Now, a key to being able to work like this is that we need to hire really good people. We need to hire the A team, or lots of A teams, and we need to fix these teams as well. Another thing we do when we tear down projects and we recreate them is tearing down all this culture and expertise that has built up in a team that you cannot recreate quickly. It takes months to build a new team and a team's dynamic. Putting together new people, you are not going to get the same results so we keep teams together. A team can just work on the next new valuable thing, whereas in a project mindset, we tear it down, come up with this new thing, and then bring together a new team. If we have cross-function teams, we can keep these teams together and they can work on anything.

Providing an infrastructure that enables continuous delivery is also important to enabling this way of working.

The other thing we need to do is to enable continuous delivery. Even if we are not going to be continuously delivering into production (that is kind of a business call), we do need to enable it. So we need an infrastructure that allows teams to rapidly deliver software because that is the only way that we can get real feedback, really valuable feedback, on it.

So, even just getting it into a demo environment – somewhere where people can play with it and we can look at the product holistically and see where we want to take it next – is absolutely crucial as well,if you want to be able to work in this way. If you can't deliver software rapidly in these kinds of small chunks, then you can't work in this way.

This approach results in more predictability in costs, which allows a value-based focus.

By having fixed teams and working in these small time-box chunks, what we are doing is fixing our cost. We know how much our teams cost to run over a week or a month, so we can start focusing on value. We can start comparing things by the money they are going to bring in or the value they are going to bring to our organization or our customer. So, we can sort of review the return on investment monthly or bimonthly or whatever it might be and we can stop funding if the value diminishes.

So, the really key thing is kind of taking cost out of the equation by knowing how much your development costs. Then you can actually start having conversations about the value.

You always need to be thinking fresh about "When we go to spend another month's worth of cash, is the value we are going to generate going to be worthwhile for us?"

Obviously, working in this way enables us to respond to change – so, again, a key Agile Manifesto principle. A lot of this really just sounds like agile, right? It is in some way just reframing our Agile Manifesto and saying, "Well, this is what we actually should be doing rather than incrementing and what have you."

So, we need to be able to respond to change. Even better than that, we need to able to be pro-active. We need to beat our competitors to the market with ideas. Again, we can only do this if we work in this agile way. We need to be actually able to really, really quickly change our course and adapt and be pro-active. Drip funding in this way enables the agility. We don't know what is the final thing that we are going to build over months and years, so let's keep on giving our teams constraints, allowing them *to be creative around those constraints and to come up with ideas that we will validate or invalidate. Keep on doing that and we will get much more effective results.*

Killick contends that working in this way makes teams more predictable.

It will ironically give us predictability because software is such an unknown domain that is so unpredictable. Because software is so unpredictable, what we are going to do is to place arbitrary constraints around that and try to create false certainty of the future. We say what we are going to do in a six-month or 12-month project because it gives us a false sense of security that we know what is happening. But, actually, that does not give us predictability, as we all know. We go off schedule and things change, expectations change. So, that predictability is not really there; it is just an illusion.

Whereas, actually delivering things constantly is predictability. We can work on features when actually asked for rather than putting them on a product backlog where they might never be seen again. I work with somebody who used to call the backlog the "place where requirements go to die". He knew that he would never get anything if we said, "It is going on the product backlog." If you want a feature and you want a little bit of value, you actually get it only if we work in this way.

Similarly, we want to be able to deliver these things to our customers as soon as they are built. We do not want to build this thing and then have to go through a full, three-month release cycle before the customer will see it. We need to get it in front of them as quickly as possible. That is predictability. If you are working with your supply and you know that you are going to be delivering features every week… – you can't any more predictable than that.

He acknowledged that there is still a need to agree on price and other terms with customers.

Touching on working with customers, another key Agile Manifesto principle – "customer collaboration over contract negotiation" – gets completely ignored in agile teams I see. We still start off with these horrible, fixed-price, rigid contracts that do not allow us to be agile at all. We should actually choose trust over paranoia.

This traditional idea of contracts is all about things that are going to go wrong and we need to cover ourselves and it becomes detailed – all these detailed clauses about what if this happens, what if that happens. And it is all based in paranoia.

If we look at it from another point of view and actually build trusting relationships with our customer and say, "You know what? We actually want to build things that are going to delight you and we need you to have that flexibility to change your mind. We want you to change your mind because that is actually going to allow us to build the thing you want now, not the thing you wanted six months ago."

So, we need be able to welcome and embrace change. If we start with a fixed, rigid contract, we cannot actually welcome change. It is kind of like "Oh, what if we do this brilliant new idea?" "Oh, we can't really because we are releasing in a month and the customer is expecting another thing." Well, you know what? The customer has employed us as a supplier of software because we know what we are doing. We actually deliver software. They want our expertise. They do not want to just say to us, "Do this." They actually want us to come back and say "Maybe you should do this, because this is going to give you a better result."

He recommends taking an iterative pricing approach when agreeing on payment terms.

Iterative pricing allows the customers to cut the cord early if they are happy with what they have, or in situations in which, for some reason, they are not happy with the progress of the relationship or how we are working. Essentially, just give them that iterative pricing.

This is possible even if you are working in a traditional environment with traditional contracts. I did some work for Vic Roads a few years ago and we had a very traditional contract – fixed price, fixed requirements – but the actual day-to-day working relationship we had was very different and because we had actually delivered a successful project to them before, trust had built up. So, we said to them we wanted to deliver in an iterative way and they saw the benefit of doing that because they were able to change things as they went along. We need to tell our customers that this is beneficial to them; it is not just our going off and being agile on a whim.

We want to do this because we want to deliver good outcomes for our customers.

There are times when we can provide clarity on pricing, when we can truly estimate with confidence.

I mentioned before that we are often experts in what we do and we want customers to understand that. So, if we are lucky enough, we can work in a domain where we are building things that we do a lot. Because we are a web-design company, for example, we can present customers with expectations of what they are going to get and when by virtue of the fact that we do these things all the time. If I design Web sites for a living, I know how long it takes me to design a Web site of a particular level of quality.

In this kind of simple, basic, standard example, the more you pay, the more quality you are going to get – quality sort of in a subjective manner, because actually the real quality comes in what actually delights you when you start playing with this thing in front of you. But what it does allow us to do is to satisfy that need for an up-front expectation. We know how long it is going to take because we build these things all the time. We know it takes, say, two weeks to build this particular type of Web site with our fixed teams so we can price things without having to worry about estimating everything up front. We can say to the customer, "You know what? We are going to build this together and we are going to build something awesome and you can change your mind and we will change our mind and we will end up with something of the same ilk as some of the other things we have done for this amount of money."

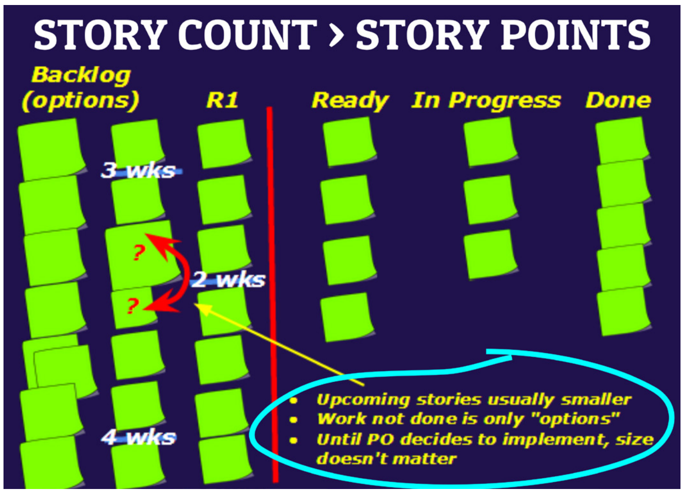
Killick then asked how we deliver features without estimating.

This is down at the team level. If we work with fixed teams that work together in a domain they know about because they've built up all this knowledge together, we can start slicing features and keep on slicing them until we are satisfied we have reached a particular level of simplicity.

So, rather than the concept of taking an epic story and breaking it up into, say, three or four stories, and then estimating each of those stories until they are all happily small – small by our estimate definition – what we can do is to come up with a heuristic. For example, each story should have one acceptance test and only one acceptance test. What that allows us to do is to slice features into stories and to stop when we hit that heuristic condition rather than stopping when we think we have small things.

That is one example of heuristic and there are obviously many other ways that you could do it. This one in particular I found very effective to create stories that end up on average taking only three or four days at most. Now, it does not matter that you are going to get outliers. You are always going to get things taking longer than other things – that is going to happen. But what you care about is that, on average, you get a certain number of things done in a week and so you can use that information. It is empirical information with which to price your features.

He presented an example of a backlog and story wall.


STORY COUNT > STORY POINTS
Backlog (options)
R1
Ready
In Progress
Done
3 wks
?
?
2 wks
4 wks
Upcoming stories usually smaller
Work not done is only "options"
Until PO decides to implement, size doesn't matter

This is something I do quite regularly to product owners. You have the team board on the right – that is, ready, in progress, and done. Then on the left hand side is kind of the product backlog area. Here we are using throughput and we are counting stories rather than using story points, because we sliced these things so we know they are roughly the same size. We do not need to drill down with any individual stories.

On average, we get, say, five things done in a week. We can start looking back and saying, "Well, you know what? Things of the sort like displacing a queue are going to take at least two weeks. Things a bit further down are going to be at least three weeks."

This information is really powerful because if a product owner says, "This thing was down at

> *number 18 in a queue. How long is this going to take?", we can answer with "It depends. Do you want to do it now, or do you want to leave it where it is in the queue? Because that is actually going to change the answer." If we leave it where it is now, all you can answer is "Well, it is going to be a few weeks before you can get that thing. If you really care about this and you want it earlier, I strongly advise you to move it up the backlog."*
>
> *Let's say we are releasing in three weeks' time and we have too many stories to get done, based on our throughput, when we want to start driving decisions by that. Again, we are using constraints to make decisions rather than just go, "Oh, that means our release date is going to be pushed out." Let's have that conversation with our product owner and say, "If you really want this thing, you need to move it up the list if you want it in this release. Otherwise, it is going to be the next release." Now, of course, if we are not working in big releases as with the approach I talked about earlier, it does not matter so much because we are constantly delivering anyway. But if you are not at that point yet and you are still working with, say, three-week or monthly releases, you need to be able to do this and make decisions based on these constraints.*
>
> *So, we can essentially derive the cost from throughput and by doing that we can have models like price per feature because we know how much our features cost on average. That is good enough to price our features with.*

He summed up the importance of enabling a culture of honesty:

> *Why is this all so important? This is really the crux of it. All the other stuff is just mechanics and getting rid of story points or what have you, but in order to build effective software, we need to build in a culture of honesty.*
>
> *The problem with estimates isn't so much the concept of estimates themselves – because they are fine if we know that they are estimates – but that the way we treat them in software is not like an estimate. They drive deadlines and they drive promises and because of this, we get all kinds of problems. Steve mentioned yesterday about gaming the system and he mentioned how story points might be gamed. I quite agree with him there: they can be gamed. If you measure people by story points, they will game the system and you will get nice release burn-up charts that make it look like everything is going great. In reality, people are constrained in working with fear rather than creativity.*

Give people the freedom to be creative and remove the stress from the environment, Killick suggests.

> *We need to give our developers and our designers the freedom of creativity to be able to actually make good choices, build the right thing for their customers. If we do not give them that, then we are always going to be making decisions based on the estimate that we made up front rather than on what is the most valuable thing for the customer that we should do next.*
>
> *And the other point I want to finish on is that I have actually seen all these extreme sides of people working in these kinds of stressful conditions. I worked with a guy who went through a divorce because he is basically held to a promise that he'd made year ago and spent too much time at work, which led to the divorce. I have seen people have heart problems caused by the stress of being in a working environment that does not allow them to be creative and not worry about constraints that are sort of binding them.*
>
> *So, even down to the level of every time someone works late and they're not at home with their family, they are not going to their kids' soccer match – these are really, really important things. It is not trivial. We laugh and joke about estimates, but there are some real, serious issues going on in this kind of culture and I personally do not want to work in that kind of culture. This is why I talk about this stuff.*

He finished with:

On time. On budget. Those are not measures of success. We need to get rid of them because there is nothing successful about delivering on time and on budget. The success comes in delivering what are our customer wants and delighting them and ourselves, as practitioners, as well. That is where we get our motivation and delight.

ABOUT THE SPEAKER

Neil Killick is an independent coach and consultant with over 17 years of experience delivering software in various capacities. He is a Certified Scrum Master (CSM), Certified Scrum Product Owner (CSPO) and Certified Scrum Professional/Practitioner (CSP). He has spoken at Swinburne University of Technology, Melbourne agile BA and scrum groups, the Limited WIP Society, and the LAST Conference.

WATCH THIS PRESENTATION ONLINE ON InfoQ

Planning and Controlling Complex Projects

by *Jutta Eckstein*

Working on large and complex agile projects for more than 10 years, I've seen planning and budgeting typically based on trying to predict how development will turn out. Very often, the development team estimates the stories but the budget for the whole project is independent from those estimates. Especially for complex projects, this leads most often to (unwanted) surprises.

Learning about Daniel Kahneman's and the Beyond Budgeting folks' work helped me a great deal in better understanding how planning, estimating, and budgeting relate and why the traditional approaches don't work.

Of course, you all know how this works in the small scale, how you plan and steer a project by iterations. But how do you decide in favor or against a project, how do you define the budget for starting a large project, and how do you know how your tiny iterations (across many feature teams) fit into the long-term project goal? Please note, I'm talking about projects with 50-300 developers that take, for example, three to five years to finish, or comparable large-product (line) development.

Prediction is not possible

Daniel Kahneman, a psychologist and winner of the Nobel prize in economics, has concluded that most everything is based on coincidence. One of the examples he uses refers to history: chances stood 50:50 that the zygote that became Adolf Hitler would have been female. Who knows how this would have changed the world? In this way, predicting complex events is not possible.

Kahneman also collected work from colleagues to verify his point. For example, Philip Tetlock, a psychologist at the University of Pennsylvania, collected more than 80,000 political and economic predictions of people who make a living predicting the future – real experts! Their predictions were worse than applying a normal curve of distribution to the situations. Yet, when proved wrong, hardly any of these experts acknowledged that their predictions were wrong. Almost all came up with excuses or reasons why they believe they were actually right but with incorrect timing – without indicating what kind of timing would have been right.

In another study, Terry Odean, a finance professor at the University of California, Berkeley, analyzed approximately 10,000 discount brokerage accounts that participated in 163,000 trades. Discount brokerage firms do not advise investors on trades; they simply execute the trades investors ask for. In this analysis, Odean found that stocks these investors sold performed on average 3.2% better than the ones they bought. (The interesting aspect of brokerage is that for every trade, there must be someone who believes that it will be better to sell the stock and someone else who believes buying it would be better. Traders on both sides of a sale are typically regarded as experts in the field.)

When Kahneman worked for the Israel Defense Force, he created a test that he used with his group to evaluate candidates for officer. You can imagine that kind of test – in addition to interviews, candidates had to solve tough live problems in which they had to build something to get a team over another thing. In the test, it became apparent who is taking the lead, who is opposing, who is a good team player, and so on.

Kahneman and his group developed much confidence in assessing the candidates' qualifications based on observing them during this test,. However, the feedback they received from the commander every few months was that their evaluation was only a tiny bit better than a blind guess. Despite that, neither Kahneman nor his colleagues changed their approach or conclusions based on observations. It remained obvious to them that an applicant taking the lead in the test would be a good candidate for officer. They were just too convinced about their impression that changing the conclusions seemed to be impossible.

So what can we learn from those studies? There are two important lessons to be learned:

First, predictions will always be error-prone, just because the world – or any complex event – is not predictable.

Second, high subjective confidence (as experts often show whether recommending soldiers for an officer career or estimating a project) is no sign of accuracy. It is only a sign that the expert has a coherent story. High confidence (sometimes called "intuition") can only be trusted if you are acting in a stable environment.

There is also a third lesson in Kahneman's research that is the foundation of iterative development: unlike long-term trends, we can predict short-term ones with fair accuracy as long as they are based on previous behaviors, patterns, and achievements.

Beyond Budgeting to the rescue

Beyond Budgeting is a development driven by CFOs from different companies. They were unhappy with the effect budgeting had on the success of the respective company. The typical experiences – and you might be able to relate to those – are twofold:

Imagine somebody asks for a specific budget for a project that then turns out to cost less than predicted. Fearing that future budgets will be restricted based on the current one, the project team spends its surplus, although not for the original intended purposes. This isn't really contributing to the company's success.

Now imagine that someone asks for a specific budget, but market changes mean the project comes to need twice the expected budget. Because the team did not ask for the newly required funds ahead of time and therefore it's not possible for them and the company to act according to the market needs. And again the company's success suffers from the original strict budgeting.

Based on these experiences, the Beyond Budgeting people developed different principles and recommendations. They don't necessarily mean to get rid of budgets but to make them more flexible. For example, to overcome inflexible annual negotiations on the budget, the recommendation is to use a rolling budget, which means to verify every month where the money is best spent. An alternative to the rolling budget is event-based budgeting that allows a budget to be reconsidered whenever changes occur.

One of the most important insights of Beyond Budgeting is to differentiate between target and forecast. The rationale is that every goal should be ambitious whereas the forecast (or estimate) is a way to close the gap to the goal. Now if both target and forecast are forced into one number, either the target isn't ambitious enough or the estimate is a deception.

Applying Beyond Budgeting

The help to decide in favor or against a project or product, a senior developer or even a team works to estimate the required effort and therefore cost. The problem is that at that point in time, not much is known about the project and therefore the estimates are not really solid. Sometimes sales comes up with an estimate without consulting development and upon which the decision is made. (And, of course, no matter who comes up with this first estimate, the development team is always assured that this number will not be taken seriously and can be adjusted once we know more....)

Given the findings of both Kahneman and Beyond Budgeting, this approach isn't helpful. First of all, it ignores Beyond Budgeting's differentiation between target and forecast. Second, in contrary to Kahneman's insights, the approach assumes that complex projects can be predicted. Third, it runs counter to Beyond Budgeting's advice against approving the budget up front.

What we need instead is first to clarify the overall goal. In order to not rely just on one group of experts (remember Kahneman's findings), I recommend performing this clarification in a diverse group. Depending on the project, this group should be composed of representatives of the customer, marketing, sales, and product management, and should be supported with scientific methods in order to find out the real needs of the market, as the principles of lean startup suggest. The outcome of this clarification is not an estimate but the definition of the business value that the company (or the customer) expects to achieve with the project. Looking at the business value shifts the mindset from what it will cost to what we will gain.

The business value can be defined through discussion using a Delphi session or, as I have done in the past as well, by using a variation of planning poker. For the latter, instead of using estimate numbers to drive the discussion, the group members use numbers referring to the business value.

Instead of asking how long a project will take, the same diverse group has to find out how much they are willing to spend (based on the business value they previously came up with). Naturally, this group will struggle (at first) to come up with a number for the investment just as developers do when coming up with their estimate. Yet, the business has to decide where it wants to spend money. From the development point of view, everything can be made in a highly sophisticated and expensive way (which typically is in line with the working ethics, speaking for example of clean code) – but at the same time things can also be implemented at low cost (which could lead to unmaintainable code). The business has to decide how much the business value is worth (which can result in asking for a cheap solution).

This does not mean that at this point there is no responsibility left for the developers in terms of planning; their task is to ensure business understands what impact a cheap (or expensive) solution might have. Consequently, estimation is not in the focus at that point. It is the decision on the business value and the investment the company wants to make that has to drive the development. Again, these drivers are created by the business and not by development. Some teams do the opposite, asking development not to estimate on the story level only but also on the epic level (by estimating for example in T-shirt sizes) or on the project level. Estimating on a more coarse-grained level is not a problem per se, as long as the estimate is used to understand the content better. Yet, trying to answer how much the project, epic, or story costs (or how long it takes) focuses on the wrong question. Again, the appropriate question to ask is how much is this worth and how much are we willing to invest in it.

This way, both the business value and the investment define a framework for steering the development. At best, both should be broken down to the story level, and if not, then at least to the epic level. We have had success planning on at least three levels. The highest level is the overall project plan (often called the roadmap). The next level is the release plan, with a release lasting three months at most. Thus, before starting a release, the epics for this release will be defined in terms of business value and investment. At the lowest level, the level of iterations, we define the business value and investment for the stories before the classic iteration planning (i.e. sprint planning one and two).

Depending on the lengths of the release (and on the complexity of the project), we sometimes need what Mike Cohn once called a "rolling lookahead plan". With this, we look into the business value and investment of the stories not only for the upcoming iteration but also for the next two or even three iterations.

Planning on these different levels allows the business to define the business value and investment first at a coarse-grained level and then iteratively making it more fine-grained. Defining it on a fine-grained level right away is for large projects on the one hand impossible and one the other hand not useful because it is very likely that the business value (and the reasons for the investment) will change over time.

Applying this approach at the different levels ensures that development is always driven by business value and investment and not by estimation. I still recommend conducting a session of planning poker for estimating the stories as a development team to spread knowledge about the stories within the team but not to drive development. The resulting estimates are only important for driving the conversation, especially when team members at first disagree over the numbers. Discussion of their different assumptions will create a shared understanding of the story. This way, the estimation is still helpful for the team but it will not drive the planning.

The prioritization and the definition what is in and what is out of scope of development is based on what the story is worth in terms of business value and investment (and not in terms of how long it will take or of how many story points it costs). The business value together with investment provides a guideline for the prioritization and for the development. If a specific story has been assigned a low investment and doesn't provide a high business value, the priority of the story has to be questioned (independent of the story points estimated for it). Furthermore, it should be obvious that for such a story the development team should look for the cheapest solution and if there is none then the story has to be exchanged for one that provides a higher business value (and/or the company should decide to increase investment).

As well, the business value and the investment have to be verified regularly, similarly to the rolling or event-based budget. At least after every other iteration, look at what has been learned from the stakeholders (e.g. what kind of changes will provide a competitive advantage), from the market (e.g. where do we need to invest in order to create more business value), and from development (e.g. what advantage does a specific technology provide). Then analyze how the lessons influence the business value and the investment on the different levels. It is helpful to first take a look at what this means for the release plan – often it affects only this level, so there is no need to take it to the overall project level. Yet, at some times this process has such an impact that the changes will influence the roadmap. Certainly, when defining the business value and investment for the upcoming iteration (and, if applicable, for the rolling look-ahead plan, i.e. for the next two or three iterations) the new lessons learned should be taken into account.

In general, if the customer's competitive advantage is really our focal point then the business has to be driven by the business value. Thus far in agile development, we have been talking about the business value but we just considered the priorities (often additionally under consideration of the estimates). Only in combination with the investment does this ensure that we always maximize the value in the customer's interest.

Conclusion

Recent findings in research together with the insights from Beyond Budgeting prove what many of us have experienced: accurate forecasts aren't possible because the world is not predictable. An expert prediction with high confidence just means that the expert has a coherent story, not that the accuracy of the prediction is high. Instead of relying on an expert, ask a diverse group of people.

To avoid the trap of mixing estimation and planning, define a business value and come up with an investment you are willing to put into this undertaking and use these values for planning. These values help you to decide for or against starting a project and help you to come up with a roadmap for the project (or product) and a release plan. Especially on the project and on the epic level will the business value and the defined investment drive development.

Short-term predictions are possible, so measure the velocity and take this into account when planning the next iteration. The feedback of the iteration and of the stakeholders helps to improve the handling of both the business value and the investment. On the one hand, the business value and investment steer the iteration, and on the other hand, the feedback of the iteration helps improve the business value and the investment. In other words, the roadmap influences the release plan, which influences the iteration and vice versa – the result of the iteration feeds back into the release plan and in turn into the roadmap. The business value and the investment are treated as a rolling budget that that you regularly revisit to consider all lessons learned.

Further reading

Daniel Kahneman (2011). Thinking, Fast and Slow. Penguin Books

Bjarte Bogsnes (2008). Implementing Beyond Budgeting. John Wiley & Sons

Beyond Budgeting Roundtable (Homepage of Beyond Budgeting):

Mike Cohn (2005). Agile Estimating and Planning. Prentice Hall

ABOUT THE AUTHOR

Jutta Eckstein is an independent coach, consultant, and trainer from Braunschweig, Germany. Her know-how in agile processes is based on over 15 years' experience in project and product development. Her focus is on enabling agile development on the organizational level. She has helped many teams and organizations all over the world to make the transition to an agile approach. She has a unique experience in applying agile processes within medium-sized to large distributed mission-critical projects. This is also the topic of her books, Agile Software Development in the Large and Agile Software Development with Distributed Teams. She is a member of the Agile Alliance and a member of the program committee of many different European and American conferences in the area of agile development, object orientation, and patterns.

READ THIS ARTICLE ONLINE ON InfoQ