ThoughtWorks®

PERSPECTIVES

How do you estimate on an Agile project?

Exploring common approaches and their adaptations from real-world projects

Share this ebook.

Contents

Estimation can be a difficult beast to deal with; more so on an Agile project. How do you estimate when you don't have a list of requirements that is complete or signed-off by the customer? Or a nailed-down schedule? What should your currency of estimation be? How do you estimate on a distributed team? Is it worth estimating at all?

Let's analyze these questions, starting with the basics

Why do we estimate?

“Purpose of Estimation”

An analysis of the reasons why we estimate to improve our estimation efforts

Martin, **Chief Scientist**

My first encounter with agile software development was working with Kent Beck at the dawn of Extreme Programming. One of the things that impressed me about that project was the way we went about planning. This included an approach to estimating which was both lightweight yet more effective than what I'd seen before. Over a decade has now passed, and now there is an argument amongst experienced agilsts about whether estimation is worth doing at all, or indeed is actively harmful. I think that to answer this question we have to look to what purpose the estimates will be used for.

A common scenario runs like this:

- Developers are asked for (or given) estimates for upcoming work. People are optimists, so these estimates tend to be too low, even without pressure to make them low (and there's usually at least some implicit pressure)
- These tasks and estimates are turned into release plans tracked with burn-down charts.
- Time and effort goes into monitoring progress against these plans. Everyone is upset when actuals end up being more than estimates. In effort to increase pace with the estimates, developers are told to sacrifice quality, which only makes things worse.

In this narrative, effort put into estimates is, at best, waste - since "an estimate is a guess in a clean shirt". Usually estimates end up being actively harmful as they encourage FeatureDevotion, a nasty condition where people start valuing ticking off features more than tracking the real outcome of the project.

Estimates also set expectations, and since estimates are usually too low, they set unrealistic expectations. Any increase in time or reduction in features is then seen as a loss. Due to loss-aversion, these losses have a magnified effect.

Faced with situations like this, it's easy to see how people turn their angry glares towards estimation. This leads to an increasing notion that anyone indulging in estimating is Not a True Agilist. Critics of agile say this means that agile development is about developers going off and doing vague stuff with promises that it'll be done when its done and you'll like it.

I don't share this view of estimation as an inherently evil activity. If I'm asked if estimation is a Bad Thing my answer is the standard consultants' answer of "it depends". Whenever someone answers "it depends" the follow-up question is "upon what".

(continued...)

"Purpose of Estimation"

An analysis of the reasons why we estimate to improve our estimation efforts

(...continued)

To answer that we have to ask why we are doing estimation - as I like to say "if it's worth doing well, it's worth asking why on earth you're doing it at all".

For me, estimation is valuable when it helps you make a significant decision.

My first example of an estimation-informed decision is allocation of resources. Organizations have a mostly fixed amount of money and people, and usually there are too many worthwhile things to do. So people are faced with decisions: do we do A or B? Faced with such a decision it's useful to know how much effort (and cost) each will involve. To make sensible decisions about what to do, you need to have a feel for both the cost and the benefits.

Another example is to help with coordination. The blue team wants to release a new feature to their web site, but cannot do so until the green team builds a new service to give them crucial data. If the green team estimates they will be done in two months and the blue team estimates that it will take them a month to build the feature, then the blue team knows it's not worthwhile to start today. They can spend at least a month working on some other feature that can be released earlier.

So whenever you're thinking of asking for an estimate, you should always clarify what decision that estimate is informing. If you can't find one, or the decision isn't very significant, then that's a signal that an estimate is wasteful. When you do find a decision then knowing it focuses the estimate because the decision provides context. It should also clarify the desired precision and accuracy.

Understanding the decision may also lead you to alternative actions that may not involve an estimate. Maybe task A is so much more important than B that you don't need an estimate to put all your available energies into doing it first. Perhaps there is a way for blue team members to work with the green team to get the service built more quickly.

Similarly, tracking against a plan should also be driven by how it informs decision making. My usual comment here is that a plan acts as a baseline to help assess changes - if we want to add a new feature, how do we fit it into the FivePoundBag? Estimates can help us understand these trade-offs and thus decide how to respond to change. On a larger scale re-estimating a whole release can help us understand if the project as a whole is still the best use of our energy.

(continued...)

"Purpose of Estimation"

An analysis of the reasons why we estimate to improve our estimation efforts

(...continued)

A few years ago we had a year-long project that was cancelled after a re-estimate a couple of months in. We saw that as a success because the re-estimate suggested the project would take much longer than we had initially expected - early cancellation allowed the client to move resources to a better target.

But remember with tracking against plans that estimates have a limited shelf life. I once remember a gnarly project manager say that plans and estimates were like a lettuce, good for a couple of days, rather wilty after a week, and unrecognizable after a couple of months.

Many teams find that estimation provides a useful forcing function to get team members to talk to each other. Estimation meetings can help get better understanding of various ways to implement upcoming stories, future architectural directions, and design problems in the code base. In this case any output estimation numbers may be unimportant. There are many ways such conversations can happen, but estimation discussions can be introduced if these kinds of conversations aren't happening. Conversely if you're thinking of stopping estimation, you need to ensure that any useful conversation during estimation still continues elsewhere.

Go to any conference with agile leanings and you'll hear talks of teams that work effectively without estimation. Often this works because they, and their customers, understand that making estimates isn't going to affect significant decisions. An example is a small team working closely with business. If the broader business is happy with allocating some people to that business unit, then work can be carried out in priority order; often this is helped by the team breaking down work into small enough units. A team's level in the agile fluency model plays a big role here. As teams progress they first struggle with estimation, then can get quite good at it, and then reach a point where they often don't need it.

Estimation is neither good or bad. If you can work effectively without estimation, then go ahead and do without it. If you think you need some estimates, then make sure you understand their role in decision making. If they are going to affect significant decisions then go ahead and make the best estimates you can. Above all be wary of anyone who tells you they are always needed, or never needed. Any arguments about use of estimation always defer to the agile principle that you should decide what are the right techniques for your particular context.

(Originally published at http://martinfowler.com/bliki/PurposeOfEstimation.html)

How do we estimate?

"All about points"

101 on Story Points

Anand, **PM, BA, Agile coach**

What is a Story Point ?

It is a subjective unit of estimation used by Agile teams to estimate User Stories.

What does a Story Point represent ?

They represent the amount of effort required to implement a user story. Some agilists argue that it is a measure of complexity, but that is only true if the complexity or risk involved in implementing a user story translates into the effort involved in implementing it.

What is included within a Story Point estimate ?

It includes the amount of effort required to get the story done. This should ideally include both the development and testing effort to implement a story in a production-like environment.

Why are Story Points better than estimating in hours or days ?

Story point estimation is done using relative sizing by comparing one story with a sample set of perviously sized stories. Relative sizing across stories tends to be much more accurate over a larger sample, than trying to estimate each individual story for the effort involved.

As an analogy, it is much easier to say that Delhi to Bangalore is twice the distance of Mumbai to Bangalore than saying that the distance from Delhi to Bangalore is 2061 kms.

Teams are able to estimate much more quickly without spending too much time in nailing down the exact number of hours or days required to finish a user story.

How do we estimate in points ?

The most common way is to categorize them into 1, 2, 4, 8, 16 points and so on. Some teams prefer to use the Fibonacci series (1, 2, 3, 5, 8). Once the stories are ready, the team can start sizing the first card it considers to be of a "smaller" complexity.

For example, a team might assign the "Login user" story 2 points and then put 4 points for a "customer search" story, as it probably involves double the effort to implement than the "Login user" story. This exercise is continued till all stories have a story point attached to them.

Who should be involved in Story Point estimation ?

The team who is responsible for getting a story done should ideally be part of the estimation. The team's QAs should be part of the estimation exercise, and should call out if the story has additional testing effort involved.

For example supporting a customer search screen on 2 new browsers might be a 1 point development effort but a lot more from a testing perspective. QAs should call this out and size the story to reflect the adequate testing effort.

(continued...)

"All about points"

101 on Story Points

(...continued)

Should we do a best, likely, worst case estimate even when we are estimating in points ?

This can be done by providing 3 different point values for the best, likely and worst case scenarios. It is quite effective when estimating a large sample set of stories especially during the first release of the project where little code has been written. Doing this provides a range across which estimates may vary depending on outcomes of certain assumptions made. For example a best case estimate for the Login story could be 2 points assuming integration with a local LDAP server, but if that assumption changes to a 3rd party provider integration, the worst case could be 8 points.

How do we plan/schedule a project using Story Points ?

To do that, the team needs to calculate their velocity in terms of number of points the team can deliver in an iteration. This is typically done using yesterday's weather by averaging the velocity achieved by the team in the last 3 iterations.

If the team is starting afresh, then a raw velocity exercise is done, where the team decides how many stories it can finish in an iteration. This is done by repeatedly picking different sample sets of (previously-sized) stories which can be done within an iteration. Average the total points across different picks to get the team's iteration velocity.

For example, if the result of 3 picks was 6, 8 and 10 points for a 2 week iteration then (10+8+6)/3 = 8 points is the raw velocity for the team for 2 weeks. A schedule can then be laid out assuming the team finishes 8 points in a 2 week iteration.

Can Story Points be standardized across various teams ?

Different teams will have different measures of story points based on the set of stories they are sizing. Unless they are building the same system, the effort required to finish a 1-point story by team A will differ from that required by team B in their system. This difference will reflect in the velocities of teams A and B.

If there is a large program of work split amongst multiple teams, it is tempting to attempt to standardize the point scale across these teams. This defeats the purpose of estimating using story points and it being a unit of measure subjective to a team.

How do we estimate spike stories in points ?

Spike stories are played to better understand how to implement a particular feature, or as a proof of concept. Since in a spike very little is known about the amount of effort involved, it is typically time boxed with an outcome that the team can agree upon. This can be approximately converted into points by looking at the velocity trend.

(continued...)

"All about points"

101 on Story Points

(...continued)

For example, if it is required to plan a week-long spike, and the team velocity is 16 points, then we can assign 8 points to the spike story.

Is there a way we can calculate cost per point ?
Cost per point will typically be (Cost of an iteration) / (Velocity per iteration (in points)). In cases where there is an additional stabilization sprint or regression iteration, the cost of that iteration should also be included.

Are story points an excuse for teams not being able to estimate correctly in days/hours?
The effort and time required to arrive at an accurate number in days/hours for a story weighs against the benefits of estimating as such. Moreover estimating in days/hours puts undue pressure on the team to deliver within those number of days, leading to the team unable to reach a sustainable pace, and possibly burning out .

Do Story Points relate to Business Value ?
Story points are an internal measure of effort involved in implementing a user story. It does not, in any way, reflect the amount of business value a user story provides. There might be cases where a 1-point story might provide a lot of business value versus a 4-point story in the same system. Business value is best left for the product owner and business stakeholders to be able to determine.

How do we know if the team is getting better at estimation when it is estimating in points ?
It is a popular belief that if the team were to estimate in ideal days, then it would be much easier to track if the estimation is accurate, by checking the actual days elapsed on a story and the progress against it. This is however counter-productive as the team spends hours to estimate few stories to arrive at the magic number of days before being pressurized to deliver on that magic number.

When a team is relatively sizing stories in points, a trend slowly starts emerging where similarly sized stories start showing similar time to implement them. If there is a bad estimate, then that bubbles up automatically as an exception

Should developers change their story point estimation as they learn more about the system they are building ?
If a story A was classified in the 2 points bucket, a similar story B coming in months later should be classified in the same bucket. If the team has learnt more about implementing them between when story A and story B were played, this will show up as an increase in velocity of the team.

It is good to setup a "relative sizing triangulation board" for the team with placeholder stories from the initial estimation session, for the team to relate to while sizing a new story.

"Stop saying estimate"

The subtle but important distinction between estimating & sizing

JK, **PM, BA, Agile coach**

There is a certain connotation with the word estimate. People think of cost and time. Think about the last time a mechanic fixed your car or you hired a painter to put a fresh coat of paint on the third floor windows. You are thinking about time and cost aren't you? When we start thinking about a software project we are still estimating the cost and time, but of the project not the stories. We are doing ourselves a disservice when we say we estimate our stories.

We have been relatively sizing stories on projects for years. We are not estimating our stories. When we look at a group of stories it is pretty easy to compare them relatively to each other and have a good understanding of which ones are similar. The hard part is to predict how fast we will finish each story. It is the velocity at which we complete stories, combined with the total number of points, which allows us to estimate the project's cost and length.

So why does it matter if we size our stories or estimate them?

As soon as we talk about estimating stories the client (and team) naturally start to think about the time it will take to complete a story and how much that story costs. This leads to people wanting to change the number of points associated to a story because "*it is taking longer than the estimate*". Just because a story has taken longer than anticipated, does that mean its relative size is different to all the other stories? Maybe... but most often it is that our velocity is not what we initially thought it would be. Talking about the size of the story helps to focus on the velocity being the value that is different from expectations rather than the number of points on the story.

With this mind set, we can move away from constantly updating the the number of points on every story and instead focus on improving our velocity by finding ways to be more effective and reducing waste.

"The Bucket Theory"

A fruity analogy to relative sizing

Malcolm, **Principal Consultant**

Over a year of presenting agile fundamentals to teams has taught me that the topic of estimation seems to strike fear and horror into people. The process of estimating seems to go something like this:

1. Pick a story
2. See the code base
3. Talk about how you would implement the story
4. Get the BA and beat them for exact details of how the solution should look
5. Get the QA and beat them for exact details of how they are going to test it
6. Wring hands for a bit. Panic a little
7. Assume because it's in component A and Jeff is the component A guy, and he is pretty quick - maybe a few days?
8. Stick a finger in the air and say "Well 3 days X 8 hrs. is 24 so 24 hrs.!

Alright, maybe its not quite as painful as that for all teams but what if I told you that on at least one project I worked on we estimated a years worth of stories in a few days without actually knowing too much about them? In itself not that remarkable, After all we could have just randomly assigned numbers to them, but the really remarkable bit is we delivered almost exactly to schedule!

So, how should true agile estimation be done?

Focus on people? On your team, pick who you think is the best developer (A) and the worst (B). Now pick a story. How many hours would it take A to deliver it and how many hours would it take B? Do we put two estimates on it? If yes, how do we forecast work? If no, how do we deal with the disparity between who is doing it? This often leads to last-minute estimation during iteration planning based on who is going to actually implement the stories – which short changes the business.

Or on time? Story 125 might be interesting to do if it takes 3 days, but not if it takes 6. So immediately we have a problem with estimating in time. Though it is seductive and all teams try it for a bit.

Or a better way? Let's say I am a voracious eater of fruit and let's say my mate is slower than me. If it takes me 2 minutes to eat an apple, it takes him 4. Let's suppose delivering your project is similar to us trying to eat all the fruit in a bowl. The pieces of fruit represent stories and I have some plums, apples, bananas, mangos and some melons.

I know a plum is about half the size of an apple and a banana takes about as long to eat as a plum. A mango will take me about twice as long as the apple and the melons about 4 times.

So I can allocate them into buckets that represent the relative size and complexity of the fruit:

(continued...)

"The Bucket Theory"

A fruity analogy to relative sizing

(...continued)

Plum/Banana=1, Apple=2, Mango=4, Cantaloupe=8

But I hear you say – "Surely this is just working out relative size using time so why not just use time?" Because relative size remains the same no matter who eats the fruit. My mate who takes 4 minutes to eat the apple will take 2 minutes to eat a banana and 32 minutes to eat the cantaloupe *but the relative size stays the same*.

This approach is often easier, as teams can look at two stories and very quickly see one is half as complex or twice as complex as another. Without all the painful processing at the beginning! I have seen teams do this with hundreds of stories in a matter of hours. So, we now have a unit of size we can agree on, we'll call them Story Points.

"But that doesn't change the speed you and your mate eat fruit, right?" Well, this is the best bit, it *doesn't matter* what speed the members of the team work at. If I break our fruit-eating task into iterations of 10 minutes, my mate and I can eat 15 points worth of fruit per iteration (I eat 1 point/minute and he eats 1 point/2 minutes) We'll call this 15 our "team velocity". As this looks at the team as a whole, it thus absorbs differences in speed between developers. In agile as always it's all about the team! (If you are trying to work out individual velocities, stop immediately – it is the path to madness and creates underperforming teams.)

So let's now take a look at the fruit bowl:

12 plums x 1 = 12 points
6 bananas x 1 = 6 points
6 apples x 2 = 12 points
3 mangos x 4 = 12 points
2 cantaloupes x 8 = 16 Points

The scope of our fruit bowl is 58, which isn't divisible by 15 (our team velocity). As iterations are a bit like movie tickets (i.e. it doesn't make sense to buy 0.866666 movie tickets) we round it to 4. We should thus have eaten all the fruit in the bowl in 40 minutes - our estimated release date.

This now empowers our product owner to work with the scope to adjust the release date. Add 6 apples and the delivery gets pushed by an iteration or remove both cantaloupes and deliver an iteration earlier, or take out all the bananas and 10 plums and add two more cantaloupes and we should deliver at the same time (though probably with an abiding hatred of cantaloupes).

(continued...)

"The Bucket Theory"

A fruity analogy to relative sizing

(...continued)

Enough with the fruit!

How do we apply these techniques to our projects?

- To start with, always estimate stories against each other, not individually. We thus need to have a frame of reference, to relatively "size" the stories. Pick a story that feels smallish (but not the smallest) and call it a 2. Use this as a reference.
- Relatively size each story against the bench mark story by discussing only the implementation details that *affect its size*. For e.g., if the decision to use SQL Server vs. Oracle doesn't alter the story's relative size, don't discuss it. Capture all assumptions.
- Put each story into a bucket 1, 2, 4, 8 or 16.
- Try to keep your story size small. Ideally an 8 should be able to be delivered in one iteration. Anything bigger, and it's best to use an "epic" to indicate that it needs to be broken down.
- For each story bucket, do a quick review of the stories in them and their estimates. Are they all reasonably close in "size" to each other? Shift buckets if required.
- Then work out your current understood scope just by adding up the numbers.

FAQs:

Why these buckets? What if we have a 100 point story?

It goes into the 16 Bucket. The 16 bucket is also a catch-all bucket for stories too big or too poorly understood to estimate.

Why do we have such strict limits on the sizes of stories?

Primarily for the sake of accuracy. I can be reasonably accurate relatively sizing the fruit in the fruit bowl but once you start asking me "How many apples in the empire states building" I pretty much have no idea. So having stories that don't size isn't useful. They can't be delivered in an iteration and inevitably turn into mini waterfall projects.

But, surely some of the estimates will be wrong?

That's true, but once you have your estimates don't re-estimate the stories. There is a simple reason for this. On most projects you'll have a couple of 2 pointers that turn out to be 8's and a couple of 8's that turn out to be 2's.

What happens when we permit re-estimation is probably pretty obvious – The 2 pointers become 8's and the 8's ..... stay 8's. So by not allowing re-estimation we pay back the debt of the bigger than expected 2 with the smaller than expected 8 and everything just kind of works.

"Using points is *not* the point"

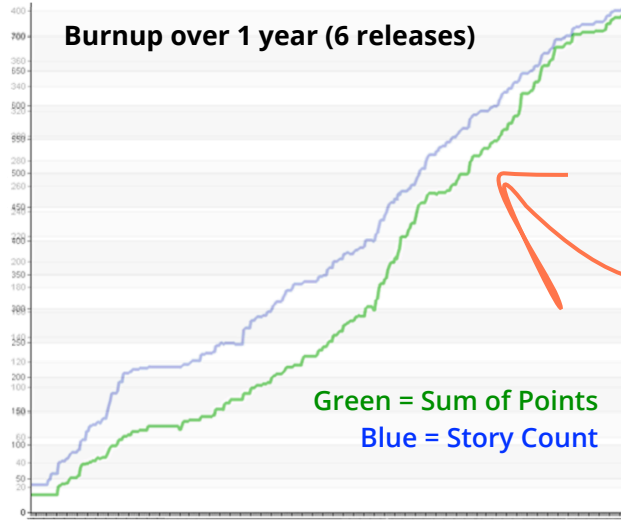
An analytical argument on why estimation shouldn't be focused on points

Juliano, **Agile coach, Delivery Lead**

I have done 3 projects in a row where we did not use story points and simply counted stories. I'm a big advocate of that approach. Let me explain why.

I'm an estimation geek who loves the nuances of estimation, and have used function points, use case points, COCOMO, and story points for over 10 years. Over time, I've become convinced that the *more we estimate* past the very initial point, the *less accurate* we get. Additionally, long-drawn, "scientific" estimation exercises generate wrong expectations of certainty. Does this sound familiar?

PM: Your original scope was 100 points, but now it went up to 130 points, so you have to cut 30 to deliver the release in the original timeline.
Sponsor: But the scope is the same. We have exactly the same 30 stories from when we started!



PM: Yes, but 100 points from then turned into 130, because we now know more about the complexity.
Sponsor: But it is the same scope and business objective. We haven't changed it! Hmm...I can't see how that story is a 5-pointer, it looks like a 3. Let's review all the estimates...

We all know how this story plays. Business feels tricked by our "bloating" of points (and in their perspective, not of the scope.)

On the last 3 projects, I measured average days/*point* and the standard deviation of same-sized stories. I found the spread to be very similar to calculating average days/*story* and its standard deviation. Using points did not give us more predictability.

On comparing burnup charts of the sum of points and story count, I get exactly the same insights in terms of progress.

I would also draw exactly the same kind of conversations, which is the real "point" here.

(continued...)

"Using points is *not* the point"

An analytical argument on why estimation shouldn't be focused on points

(...continued)

I now prefer to have a different conversation with clients using these two ways:

1. **Define release scope not in terms of stories but in terms of features we're delivering and their business objectives.**
   Points represent 3 types of scope:
   (1) Feature/function
   (2) Richness/usability/depth
   (3) Technical complexity
   However clients generally only consider the first one and get confused when we say "scope increased" due to the other two types, as their business objective has not changed.

2. **Discuss scope in terms of projected number of stories we think we can do (forget points) and put rules around the maximum duration of a story.**
   For example, the team should not pick any story that they think will take more than <max> days to complete. So if the "scope" (any of the 3 types) increases , the story can be split and the last one in the queue might fall out.

Not only are these conversations easier, but they also get people focused on simplifying the last two aspects of scope that don't "directly" contribute to the business objective, so they can actually get more of "their scope" (features/functions) in. And we don't get into (endless) discussions about points and sizing stories.

In Summary:

I do think that there is an evolving progression in a team's approach to estimation:

1. Estimate in effort (days, weeks etc.).
2. Estimate in points (use case / story points etc.).
3. Estimate by counting stories/cards. To ease the transition, I generally say, "Let's assume all stories are 3 points and split those that aren't", and multiply every story by 3. This helps to drive right behavior towards smaller, similar-sized work packages that give you more flow.
4. Forget estimates and simply work on continuous flow, focusing on cycle time.

Now, to get to each stage there must be some stakeholder buy-in. I have had honest conversations to the tone of "We can game the points all day long, but that won't get your business outcomes delivered, so what would you rather do?" Sometimes the answer has been that they can't help it and still want to fight points, so fight points we do until we can change it.

FAQs:

I like cycle time and features as metrics. But how do you handle that (often long) period before the former stabilizes?

There is no magic bullet. (Projected) velocity or (projected) cycle time can help you take a call as to how much you can deliver in a certain time.

(continued...)

"Using points is *not* the point"

An analytical argument on why estimation shouldn't be focused on points

(continued...)

1. If stories are diverse in size, I do the usual total points scope calculation and simulate 2-3 iterations with the Devs, asking how many stories they think they can complete; then infer velocity from there.
2. If stories vary less in size, I count the stories and ask Devs how long (Dev cycle time) it would take them to develop them. I then infer available capacity by multiplying number of available pairs (discounting capacity for tech tasks) by time available and dividing by guessed cycle time. This gives me how many stories can be completed, (need to factor in critical path/dependencies). This is also useful as a second (different) way to validate velocity.

Regardless, I always ask the client the amount of risk they see in the scope/complexity. What is their current understanding of the stories and unknown scope? How many points/stories/scope buffer do they want to add? My general rule is 0-10% is unrealistic, 20-30% is manageable, and >30% if everything is very experimental. As they give me the number and I just give a recommendation, I find it easier to have conversations later when unknowns unfold (to number of stories or points).

When planning the release I do it at epic/feature level linked to a business outcome, and then split it into stories, asking if every derived story really does contribute to it.

Thus story culling happens more naturally. When things stabilize, I have a conversation about flow and cycle time.

Everyone's advice is to have all of your stories roughly the same size but I generally see a huge spread. Have you achieved that, or are you saying it doesn't matter?

To an extent I've found that it doesn't matter. I've found that the spread goes from 0.33 day/point to 3 days/point, as the final stories tend to get easier (more certainty) than their size in points indicate. This huge spread makes me think that from a scope management perspective there is no value in discussing points, as I get roughly the same data with a count (graph on Pg 12). Without generating wrong expectations about our certainty.

If someone is really wedded to points I just say, "Multiply each story by 3 points (I anchor estimates around the "average" 3 pointer story) and the difference will come in the wash". This has worked just as fine (or as badly) as if I discussed the difference between 2 and 3 points. Also, I don't allow stories>8 points (or that the team says would take >5 days to complete) in the backlog, as that size seems to indicate amount of risk rather than complexity.

To me the value of an estimation session is to align the team around scope, solution, risk and complexity as different people discuss estimating the same story with different sizes in points; not from the actual number that comes out at the end.

"Estimating without points"

Applying Lean principles to effectively relatively size your stories

Ian, **Principal Consultant**

Points = $/£/¥/€/?

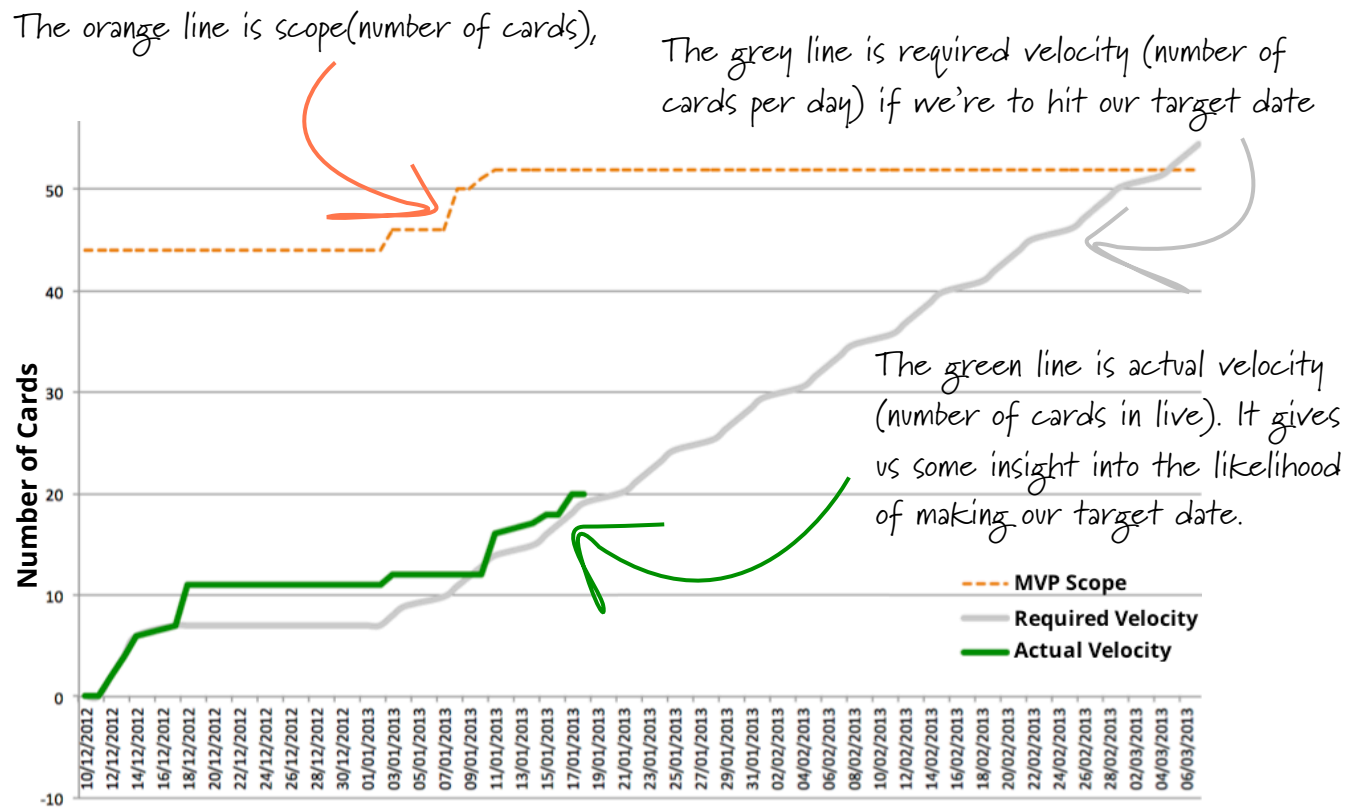
Of course they don't but I constantly see points being abused within organizations and becoming currency for staff manipulation. "The team only delivered 17 points this week when they said they'd deliver 20. Can the team work the weekend to make up the points they owe us?"

On my current project we don't use points. We simply relatively T-shirt size our stories.

This allows us to very quickly size the stories and not waste too much time on estimation. We still have a burn-up chart but it's based on the number of cards delivered to live irrespective of card size. See the burn-up chart below.

The obvious problem with this approach is the false sense of progress if you play all the small stories first – essentially saving up trouble for the future. This is where a bastardized adaptation of the Yamazumi concept comes in.

(continued...)



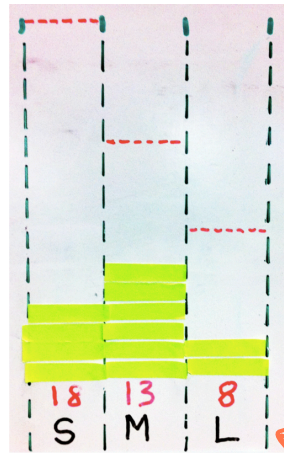
"Estimating without points"

Applying Lean principles to effectively relatively size your stories

(continued...)

Yamazumi

In Lean manufacturing there's a concept called Yamazumi. It's basically a graphical representation to aid in creating balance between operator cycle times. With balance comes reduced variation. With reduced variation comes better predictability. To ensure we get the right balance of story sizes played we created a kind of Yamazumi chart:



From this chart we can see the spread of stories in play and appreciate the balance of stories. In this example, I would be encouraging the team to play a large story next.

WARNING!

Selecting stories based on size rather than business value is wrong. In our current situation the backlog is stripped right back to Minimum Viable Product which means all of it needs to be delivered to create value. This allows us to use story size as a factor in the selection discussion.

FAQs:

What about in the scenario where you haven't engaged a supplier/partner yet and you need to know how much to budget for?

There are probably many ways to do this but here's a couple of thoughts:

- **Focus on value** – Base your business case on value and window of opportunity, i.e. how fast can you start to realize some value. What do you expect (or hope) the value / benefit will be? The cost element comes in during the tender process which is just one aspect to partner selection and will be discussed in a later blog post.
- **Decide how much you want to spend to solve your problem** – There are very few organizations that have a bottomless pit of cash. The annual budgeting cycle will provide you with a ceiling for expenditure so surely it shouldn't be too difficult to work out what percentage of the overall budget you want to allocate to the problem at hand? In Agile delivery working within a fixed (or capped) budget is totally compatible with Agile thinking.

(Originally published at http://iancarroll.com/2013/01/22/agile-planning-without-points/ and http://iancarroll.com/2013/03/25/estimating-how-much-the-indefinite-might-cost/)

In practice

"Estimating on a distributed team"

A summary of learning on estimation from 7 distributed projects

Jiangmei, **BA**

In a distributed team, do they estimate stories?

➡ If yes,

- Why do they estimate?
- What are the estimation techniques?
- How much effort is spent on estimation?

➡ If not

- What do they do instead?
- Do they face any new problems?

Why do they take a different approach? What factors drive them do it differently? What can we learn from them?

To answer these questions, we interviewed 7 distributed projects at ThoughtWorks in China. And we found that:

1. Most of the teams are still estimating stories during the iteration/feature planning. A few teams only estimate during inception. Subsequently, as the project progresses, there is no estimation for the stories, just planning by counting cards.
2. The major problems they want to solve by estimation are:
   - Derive an estimated scale for a new bucket of stories to help plan future releases.
   - Provide an estimated effort for each story to help the business prioritize better (from a ROI perspective, value vs. cost).
   - Synchronize the derived understanding of the story and its context across all distributed locations.
   - Gain confidence and build customer trust by fully understanding the business/ technical context before commitment to build.
3. For the teams who estimate stories during iteration planning:
   - They use story points and the Fibonacci sequence (1, 2, 3, 5, 8);
   - They usually use a style like "planning poker" to achieve consensus.
   - For a small team (< 20 people), usually the entire dev team is involved for the estimation; but when the team size exceeds 20, only a few dev representatives would estimate the stories.
   - They use a lot of utilities to facilitate the distributed session to raise energy and ensure focus.
4. Estimation times vary - some teams can estimate 20 cards in 20 minutes while a few teams might take more than an hour for 8-10 cards. When the sizing exercise runs too long, it becomes an annoyance rather than a helpful technique.

(continued...)

"Estimating on a distributed team"

A summary of learning on estimation from 7 distributed projects

(...continued)

5. For the teams they don't estimate often,
    - They run sessions to allow the entire team to share their understanding on the stories and identify risks in advance.
    - Their release cycle is very short. They tend to move away from time-boxed iterations and their style is more like of a continuous working flow.
    - They tend to have smaller (mid-size) stories.

Why the different approaches?

When comparing the teams that are at different ends of the spectrum, we can find that the following factors might matter:

- *The agility of the client organization*: Shorter release cycle leads to less emphasis on the number of points.
- *Team maturity*: More mature teams have more adaptive planning.
- *The mutual trust between the client and the offshore team*: Higher trust levels mean lesser estimation efforts.
- *Familiarity of domain and technical context*: Teams that are more familiar with the context, require less efforts to estimate.

In Summary

- It's vital to clarify the value of estimation, and ensure the value and purpose is shared across the whole team.
- Focus on building your customers' trust - by investing in collaboration mechanisms, delivering high-quality software, and being adaptive to changing customer demands - instead of spending all your efforts on hitting the estimated scope number.
- Evolve your approach as the team grows. In the early stage, it may be worth spending time and effort to estimate accurately. However, as the project proceeds shift the focus to developing a shared understanding and building the valuable products.
- Utilities matter! It is worth investing in good hardware (e.g. large screens, stable network providers/plans, video-conference system, well-equipped conference room, etc) to facilitate effective, ad-hoc communication between teams at different locations.

"How story counts worked for us"

The why & how of using story counts to estimate

Huimin, **BA**

For about two years now, a norm has emerged on the Mingle team: "Every story is 4 points." As a BA on our team, I quipped, "Well, that's because our BAs are particularly good at writing stories." :)... And then started digging into data to understand why.

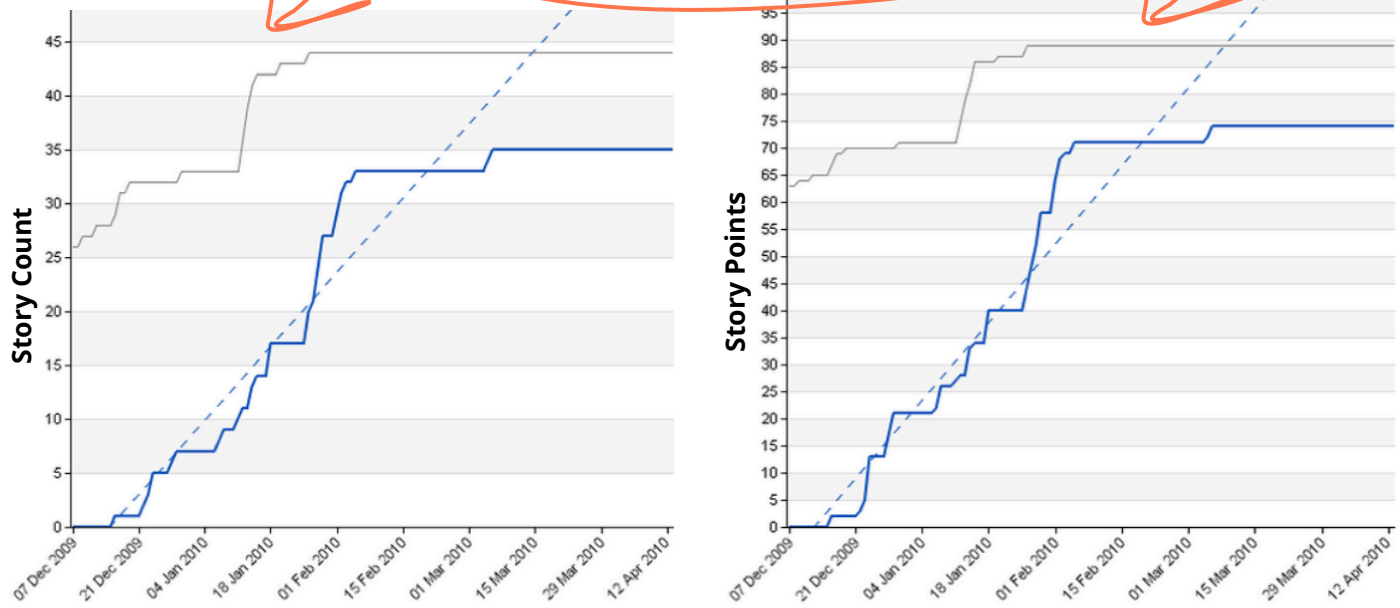
Let' s analyze our historical data

I created two charts below using data from one of Mingle's previous releases and found them to be strikingly similar.

*This chart maps the **story count** over 3 months for a release*

*This chart maps **story points** over 3 months for a release*

Aside from the Y-axis scale, can you tell any obvious difference? I bet not.



(continued...)

"How story counts worked for us"

The why & how of using story counts to estimate

(...continued)

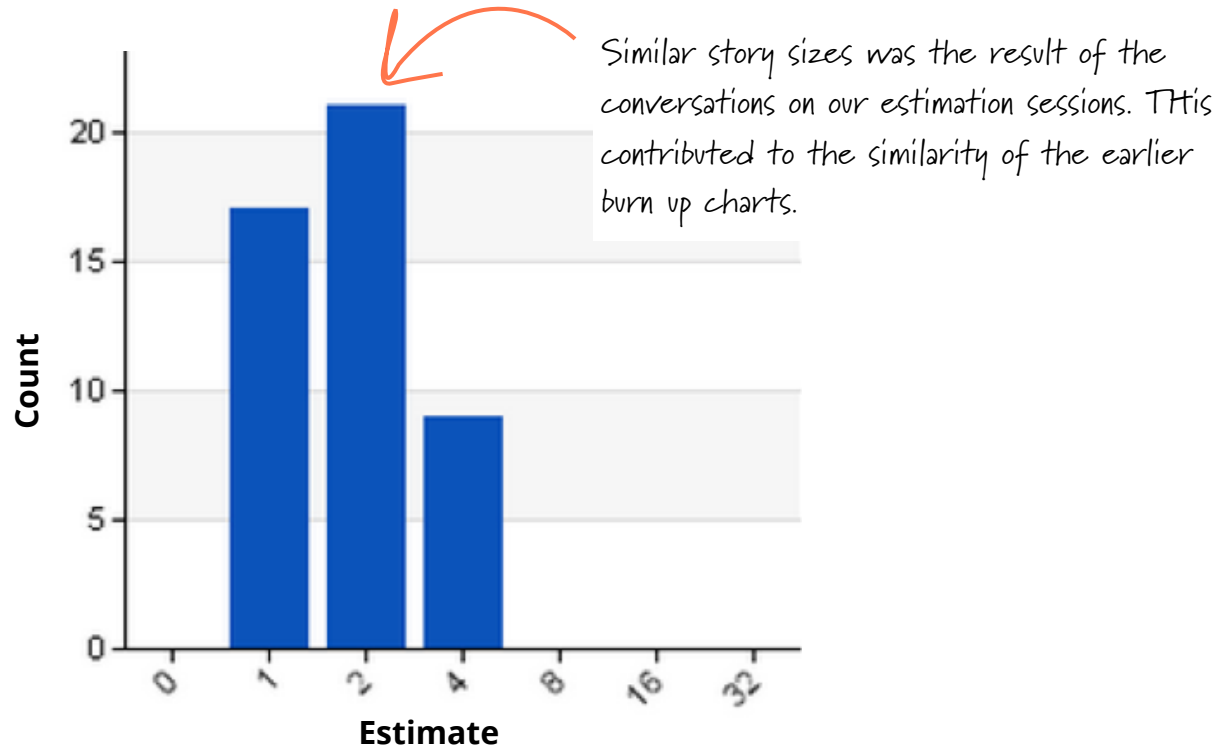
Why is that the case?

1. *Stories got broken down within the same range during team conversations*

When we estimate on the Mingle team we always have representatives of every role, if not the entire team. During estimation, everyone is involved in breaking big stories into more digestible pieces.

We use a 1-2-4-8 scale, with 8 as our threshold. Anything estimated bigger than 8 becomes a placeholder for further breaking down.

Below is the distribution of our estimates used in the burn up charts on the previous page.



(continued...)

"How story counts worked for us"

The why & how of using story counts to estimate

As you can see, additional "accuracy" that story points provide vanishes after 2 weeks. And since a forecasts 2 weeks out (regardless of it's accuracy), when there are still months of dev work left has never interested us, the point is moot.

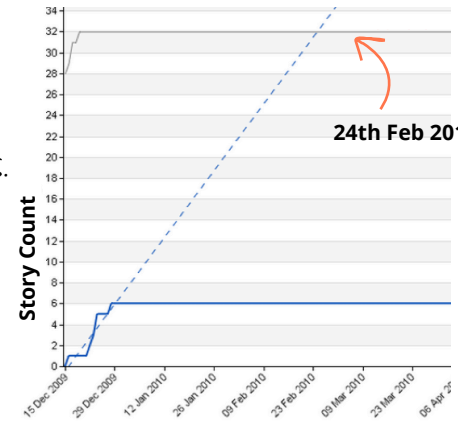
(...continued)

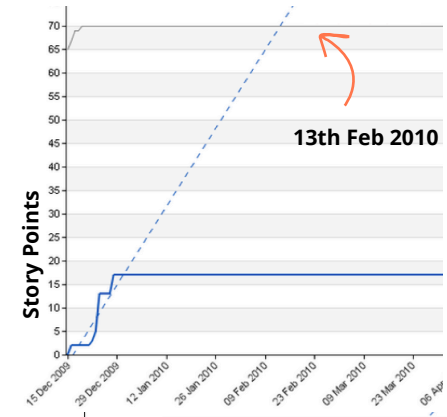
Why is that the case?

2. Size differences got evened out over time

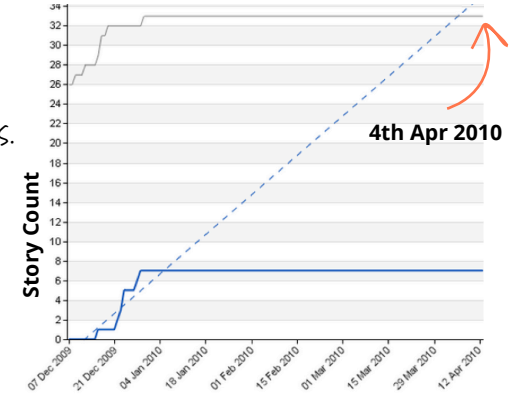
Applying normal distribution to story points, standard deviation decreases as the sample size grows.

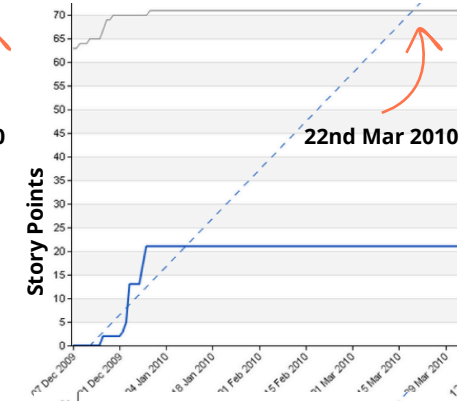
Forecast of story count vs. story point, 2 weeks out



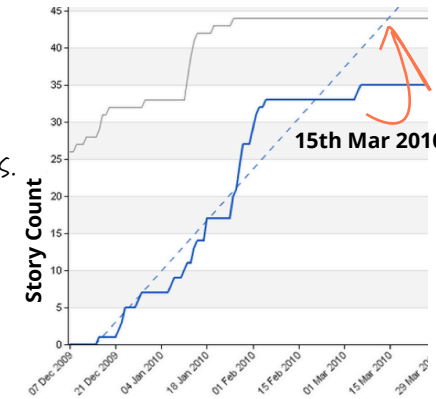


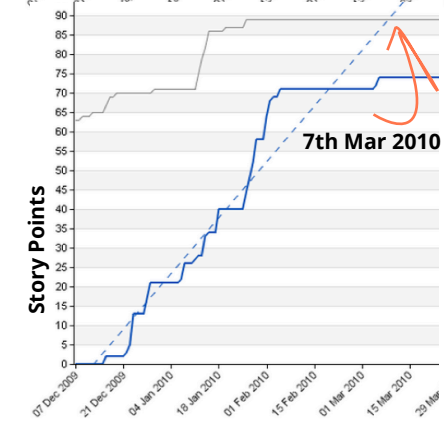
Forecast of story count vs. story point, 1 month out





Forecast of story count vs. story point, 3 months out





(continued...)

"How story counts worked for us"

The why & how of using story counts to estimate

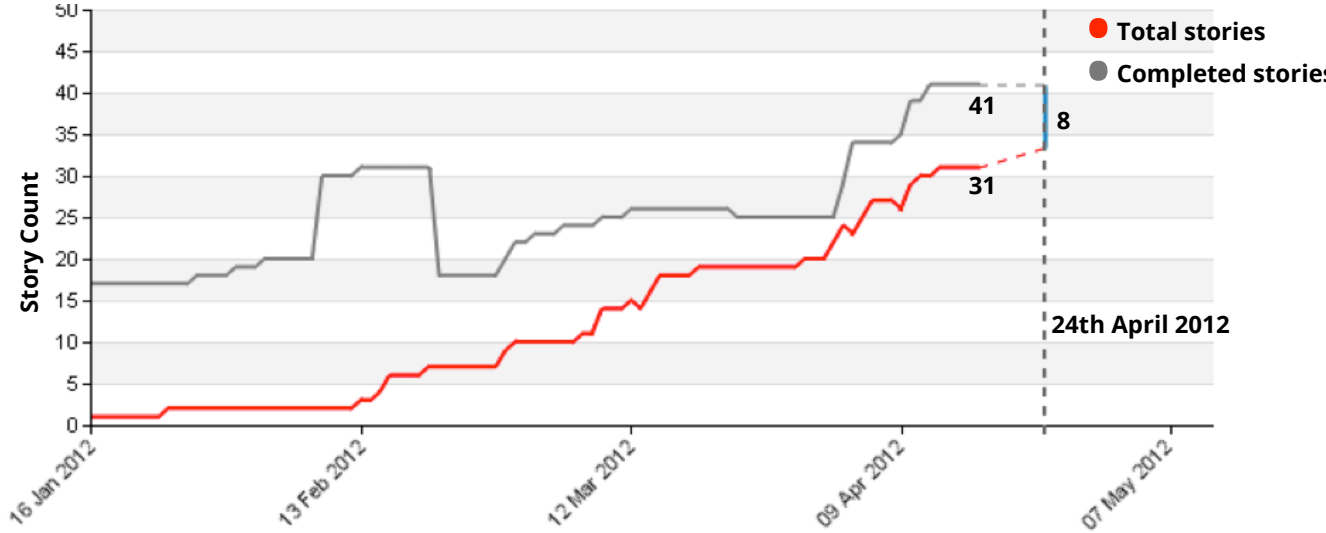
(...continued)

Which is why we refactored our process

Looking at our data, we didn't find any additional value that story points provided us (related to progress tracking). As such, we have transitioned from story points to story count:

1. We still maintain our estimation sessions. We highly value the team conversation catalyzed by gauging the size of the work.
2. Leave the estimate points as a reference on the card, which could help inform prioritization. But we do not translate those numbers into scope or capability.
3. We started using story count in our burn-up charts.

We believe that the key to progress reporting is not an "accurate" prediction, but visible signals that we can act on. We look to our burn-up chart to tell us: "Hey, it looks like we might not be able to get everything done by the expected date. Let's have a conversation."



(continued...)

"How story counts worked for us"

The why & how of using story counts to estimate

(...continued)

We are happy with this change

It has resulted in these two significant benefits:

1. *Fewer metrics, more conversations*: In estimation meetings, we have shifted focus from numbers to a collaborative conversation. This provides a better platform for our team to discuss and eventually establish a shared understanding about what to build and how. We noticed that subsequent development work became much smoother after these conversations.
2. *Less math, more effective planning:* In scope planning meetings where we used points, we had to scratch our heads to figure the exact number of points to put in or take out. Freed up from these calculations, we focus more on business value and being more responsive to ad-hoc requirements.

In summary, I would like to quote Martin to support our decision: "*So whenever you're thinking of asking for an estimate, you should always clarify what decision that estimate is informing. If you can't find one, or the decision isn't very significant, then that's a signal that an estimate is wasteful.*"

Yes, the estimation column is empty! Seeing as the estimation points had naturally phased out of our process, we had an explicit conversation during our retrospective about whether or not we should reinstitute them. We decided not to, and have been happy with it.

Select: All | None Listed below: 1 to 14 of 14. Maximize view Print Link to this page Add / remove columns

#	Name	estimate
14216	[Objective API] Events	
14215	[Objective API] Delete objective	
14214	[Objective API] Create objective	
14213	[Objective API] Update objective	
14212	[Objective API] Get list of objectives	
14211	[Objective API] Get single objective	
14193	Create card when "Add with detail"	
14185	Update Mingle logo and login screen	
14168	13.1 Help Doc	
14114	[WYSIWYG] Improve insert image experience (file upload)	
14112	[WYSIWYG] Introduce CKEditor	
14106	[Spike] WYSIWYG (step 2 - CKeditor)	
14064	[Spike] WYSIWYG (step 1)	
6459	Support ORDER BY Project_Card_Rank	

In summary

- Revisit the purpose of estimation
- Explore different ways to estimate and pick one that suits your team/project
- Understand that each team's approach to estimation evolves as the project progresses

Martin Fowler

I am an author, speaker... essentially a loud-mouthed pundit on the topic of software development. I've been working in the software industry since the mid-80's. My main interest is to understand how to design software systems, so as to maximize the productivity of development teams.

Anand Vishwanath

I work as a software consultant/ project manager/agile coach with ThoughtWorks, helping clients deliver projects using Lean and Agile practices. I am passionate about building self organizing delivery teams and am a proponent of servant leadership.

I am an PM, BA and Agile coach at ThoughtWorks. I occasionally write about process or analysis practices that I find useful. I'm currently thinking a lot on how to help others adopt agile. Outside of work, I'm an avid Arsenal fan and dabble in close-up magic. I am also passionate about good food.

JK Werner

I am an Agile coach on the ThoughtWorks Studios training team. I'm passionate about translating my experience practicing Scrum, Lean, XP and Kanban methodologies to training. I also enjoy building (& crashing) RC copters, designing build-break machines from Arduinos/nerf guns & playing rhymes on my electric guitar for my kids.

Malcolm Beaton

Juliano Bersano

I am an Agile delivery consultant and a recent ThoughtWorks Alumni. I am passionate about working with people to create awesome teams that can deliver great software products. I also enjoy traveling, reading and cooking, wine and coffee, tennis and the gym. And post-its.

Ian Carroll

I am a long-time agilist and am passionate about introducing Kanban and Agile to a number of organisations across the UK. I'm also an amateur anthropologist and am fascinated by tribes and the nature of people.

I am a BA/PM at ThoughtWorks Beijing. Having worked on quite a few distributed projects I have experiences (& interesting anecdotes) to share on working with teams in disparate time zones (standups at 7am), collaborating across countries & languages, & all the fun & madness that is distributed agile.

Jiangmei Kang

I am a Business Analyst with ThoughtWorks Studios on the Mingle. Over the course of my career, I've worked as a Quality Analyst on different software teams and as a Research Assistant at the Networking Institute. My other interests also include interaction design and visual thinking.

Huimin Li, BA

ThoughtWorks®

mingle®

Agile Project Management

Get the team together

Agile workers talk often and welcome change. Mingle creates a shared space to make quick decisions and track details, even when the team can't be together.

Share this ebook.

Be in this ebook.

Tell us your story.

We'd love to hear it.
Email us your take on estimation and if it is interesting we'll include it in this ebook

Email Us