Transcript: Mythbusting Software Estimation
Free Webinar on October 14, 2014

Hemant Elhence: Hello all, today's topic is of software estimation. I'll introduce our feature speaker, Todd Little, in a minute. Todd will cover the topic Mythbusting Software Estimation. He'll cover in this session some of the reasons why we face the challenges around software estimation, and bring a lot of real data to bear and blow away some myths and confirm some others and help us really think through this topic of software estimation, a difficult challenge for all of us in the software development fields.

For the process, we will have roughly 45 minutes or so of presentation and I'll keep inserting questions you guys have along the way. Some questions we might hold for the end, and then we'll have another 10-15 minutes or so, hopefully, to take general questions on this topic. Keep sending your questions in the Q&A panel that all of you have in here. We are recording the webinar so we'll have the recording available as well as the document that goes with it.

With that let me introduce Todd. Todd is no stranger to these webinar series. He has done a number of these webinars. Todd is the Vice-President of Product Development for IHS, a leading global provider of information, analytics, and expertise. He has been involved in most aspect of software development with a focus on commerce and software application, especially in the oil and gas exploration and production areas. Todd is also co-author of the "Declaration of Interdependence for Project Leadership". He is a founding member and past president of Agile Leadership Network. He has served on the board of directors for both Agile Alliance and Agile Leadership Network.

Many of you, if you attend Agile conferences you may have seen him in a leadership capacity there. I have seen him for a number of years there, that's how I got to know Todd. Todd is also co-author of the book "Stand Back and Deliver: Accelerating Business Agility". He has written several articles in IEEE Software and post all his publications, presentations on his website toddlittleweb.com.

With that, over to you, Todd.

Todd Little: Okay, very good. Thanks. It's good to be back here. This is a topic that is near and dear to my heart of trying to determine the challenges we have with software estimation. I'm going to go and do a little bit of exploring of some of those myths that we have around software estimation. See whether these myths are real or whether we can bust a few myths.

I asked Hemant for a large budget so I can go off and build some experiments and blow things up. He didn't come through with that so instead we're going to look at real data that researchers, some of the data that I've collected in some of my research. Quite a bit of it is from other researchers, etc., looking at the same challenging problems.

We're going to approach this with a little bit of a test first mentality here, so I'm going to start with listing off a few of the myths. Just in your mental picture, what I'd like is for you to think about each one of these as I say them, is this not a myth that we would likely just be busted? Is it one that's probably true and we can confirm it? Or is it in that grey area where all we can do is say it might be plausible? All right?

Let me just read through these quickly.

The first one: Estimation challenges are well understood by General Management, Project Management, and Teams and it is normal to be able to estimate projects to within 25% accuracy.

Number Two: Estimation accuracy significantly improves as the project progresses.

Number Three: Estimations are frequently impacted by biases and these biases can be significant.

Number Four: We're pretty good at estimating things relatively.

Number Five: Velocity/Throughput is a good tool for adjusting estimates.

Number Six: We're a bit behind, but we'll make it up in testing since most of our uncertainty was in the features.

Number Seven: Scope Creep is a major source of estimation error.

Number Eight: Having more estimators, even if they are not experts, improves the estimation accuracy.

Number Nine: Project success is determined by on-time delivery.

Number Ten: Estimation is waste.

Those are the ten myths that we're going to take a look at in varying details on each one of them. Let's get started.

Myth number one. How well do we actually understand the estimation challenges?

Okay, so estimation challenges are understood by general management. Let's start looking at that one.

Let's look at a little story of how too many software projects seem to behave, what they call Inside the Coming Storm. First off, we have a little project kick off. Little Dorothy says, “When will we get the requirements?”

Oz says, “Just give me your estimates by this afternoon.”

The team binds together. “No, we need something today.”

“Okay, then it'll take two years.”

“No, we need it sooner. I already promised the customer it'll be out in 6 months.”

Ah, yes. We're not in Kansas anymore.

The developer hero says he's going to fix this. He's just about got things under control when the reorganization hits. Then they get it done. They're almost done. They said they're done. They declare it and then they go into testing. Then what happens? Typically projects look like that.

I've done a lot of looking into projects and that's this little anecdote. When I started looking and doing a lot of research and I started going back in time and looking and came across a couple of very interesting topics and discussions around this topic.

There was a paper called “Why is Software Late” by Genuchten in 1991, published by IEEE. What I love about this paper is that he looked at surveying both the General Manager and the Project Manager as to what some of the challenges were that the teams were encountering. What I like about this is that it's looking at two different contextual perspectives, the general manager, project manager, and each of them having a different view of things.

I like to bring up the story of the importance of the context of feedback. Make sure you understand that context of who you're asking and what you're getting back. This is a little thing from World War II (WWII). In WWII, the British were doing some statistical analyses of the planes that

were coming back from the war. They were looking and seeing if they could find patterns in the planes and where they were being attacked. They're figuring that if they can figure out where they were being attacked, they can reinforce where they're being attacked and they would then have much safer planes.

A Hungarian mathematician looked at what they were doing and said, you've got the context here all wrong. You've got to remember that these are the planes that are returning from the war. The ones that got blown up are the ones that got hit in the places that these ones didn't get hit. They actually bolted ahead and reinforced the places that were not being hit and it actually worked out for their advantage. Context of feedback is very important.

Here we have two different views of feedback on this project. One from the General Manager. One from the Project Manager. Project Manager is the person down there with the team actually working. General Manager has a stakeholder but is not actively involved in the project itself. What I look at too is there are very few areas where they're in agreement.

Customer/management changes? Yes, they both agree there was some scope change and things happen. Very important issues and this does have a big impact on the project. They both agree that they probably have an unrealistic project plan. They were a little optimistic in what they had initially planned to do so yeah, that was a big challenge.

Interestingly enough, you get in the areas where then there's a disagreement between the general manager and project manager, some interesting things there. Staffing problems, the General Manager thinks, no, I pretty much gave you the people that you needed and project manager's saying, no, in fact, we kept telling you we needed more people but in fact, we never got them.

The other one, overall complexity, the general manager wasn't directly involved but this isn't terribly complicated. Project manager's thinking no, this complexity was what really hit us here with the real challenge.

Then the last one, insufficient up front planning, the general manager's thinking, oh, if only we had done more up front planning, we would have solved our problems. Project manager's saying, no, that wasn't really the problem. We did the right amount of planning. More up front planning wouldn't have actually helped us that much. A big disconnect here between how the project manager was doing things and how the general manager was doing things. Definitely challenges to a lot of our projects.

The challenge here, this quote from Upton Sinclair, is it's difficult to get a man to understand something when his salary depends upon his not understanding it.

This is an interesting summary from the investigation of the Space Shuttle Challenger incident. They looked back at what sort of estimates had been done by engineers and general management as the probability of loss of life. The engineers who looked at it had been estimating, this is before the explosion, the engineers had been estimating that it was probably somewhere in the 1-100 range. General management was looking at this and said no, this is not very large of a challenge, if at all. It's in the 1-100,000.

We all know what happened, actually, with the Space Shuttle Challenger and the overall space program, looking at the actual data there. 135 flights, 2 disasters, and 14 deaths, all sanctioned because management thought there was no chance of a serious problem. In fact, if you look at the map, the engineers were pretty close. In fact, they were potentially optimists in this case.

If you look at a software example, we consistently see a challenge with overconfidence. This was a survey done where they were asking project leaders how they were doing with success of their projects. In the early stages, in the middle of stages, they were asking this and they measured that the perception that 80% of people, 79% of people, of the project leaders, thought that they were- they perceived themselves in great shape. No problems whatsoever. Everything's green.

When they actually measured how these projects succeeded, half of that, only 42% actually had succeeded in meeting the objectives. This is a fairly common trend you see in our industry and in other industries as well, this idea of overconfidence of success, overoptimistic view of the world, of the challenges that we face.

This is a paper that I wrote in 2006 in the IEEE Software utilizing a lot of the data that I've been collecting that dated from a couple of years back from late 1990s through early 2000s. I'd collected a lot of data. We'd had a bunch of projects with a successful commercial software company. They had been doing really well during that time. Looking at it from a commercial and value creation perspective, this company was increasing market share incredibly during this time. On the success side, it was definitely there from delivering value. We had some challenges in terms of meeting some of the predicted schedule and estimation, certainly, much like other organizations.

This paper, one of the things I started with was looking at the accuracy of our initial estimate that we had for the project. If we plot this on a line here showing the initial estimate on the bottom and the actual on the left, if we had perfect estimation, this would be tracking to this magenta line. It's out of scale. It's not exactly at 45° because of slightly different scales on the chart to account to where the data plot is. What we see is that our data is pretty scattered. It's not all that close to the purple line. In general, most of the points are considerably above the purple line, which means that our actuals were over our estimates.

The interesting thing when I plotted this data, is I looked at it and said, well, I've seen something just like this. I had been reading some work by Tom DeMarco, taken a class by Tom DeMarco and saw a very similar chart and in fact, these red dots are the data that he published in his book "Controlling Software Projects", a very similar pattern.

I thought, well, there may be something here that we're seeing a fairly common pattern. It's not just unique to Tom's data or our data. I've looked at a lot of data subsequently to this and you see this data fairly consistently. This behavior-

Hemant Elhence: One question on this data. The question is, did you try to run regression on the actual data to come up with an actual conclusion on what the data is telling us about the bias of actual vs. initial estimate?

Todd Little: I'm going to go through some analysis that we did on the data and we'll see whether that answers that question and if not, we can come back to that question with more detailed responses.

Hemant Elhence: Okay, all right.

Todd Little: With my data, I had quite a bit of analysis that I did with it. I'm going to go through that a little bit more in detail as we progress. There's also some data that we've probably seen in Steve McConnell's estimation book, a very similar pattern here. Lots of data points that are above the line. We're collecting, in general, an optimism bias that we have within our industry.

If we look at it from a data analysis perspective, what we find is that 10%-20% are those that are showing under the original estimate. Only 10%-20% are actually meeting their initial estimate or beating it. Roughly 50% are at the point where it's less than 2X the original estimate, which means roughly 50% are greater than 2X the original estimate. We have to get up to the 80%-90% of projects, or actually, we have to get to 4X the

original estimate to make sure that we're covering 80%-90%, which means 10%-20% are exceeding 4X their original estimate.

Remember, this is the original estimate. This is fairly early on in the project life, pretty much at the project sanctioned point. This type of challenge, this level of gap between the 10% confidence and the 90% confidence, somewhere between 4X is a pattern that I've seen repeated and repeated in the data that I've observed. It's a pretty substantial range. It's way greater than I find most people realize that we see happen within estimation challenges, particularly in software challenges.

This is some data recently from Magne Jorgensen from Sintef in Norway. I will be quoting a lot of his work because he's done some really excellent research in the area of estimation and challenges, particularly like this project, because he actually had multiple vendors do the work and so he had some head-to-head comparison.

He put the software development project to bid out on an online marketplace called vworker.com. For that, they received a total of 16 bids. He culled this down to 6 bids that he used from vendors that had high client satisfaction. Online brokerage had satisfaction ratings so he made sure that the vendors that he chose were ones that had decent satisfaction ratings, 9.5 out of 10 scale. He had 6 vendors that he thought would be good vendors to work with. All 6 bidders, they got the bid and then they went ahead and built the software. We have a comparison of what they estimated to what they actually did and then we also have a direct head-to-head comparison of those 6 bidders.

Interesting results from this. The highest estimate was 8X below it. Pretty amazing. The actual over the estimate range, for each of the 6 people that went through it, I took some of his data and compared the actual that they did compared to the estimate that they gave. The range of this was from 0.7, so one person actually came in under their initial estimate, up to 2.9X the initial estimate. Again, if you look at that band of range, you get that very similar 4X ratio between the 0.7 and the 2.9.

Then the other thing to look at was the actual performance range. The worst performer took 18X the effort of best performer. A huge range in terms of what the estimate as well as the actual. The interesting thing about this was also that the lowest bidder was the one that actually was able to come in under their initial estimate, so not only were they the lowest bidder, they were actually by far, the lowest actual effort as well.

Hemant Elhence: One question to clarify the data. Was this data for individual kind of work, or project team-based work? The initial estimate to be done by a team or by individuals?

Todd Little: I don't know whether this was distinguished as a team activity or an individual. It was put out to, I think it could be either. I don't know how large the projects were so I don't know whether this was an individual performance. It was actually a delivery of a system. It could have been done by a team. I think this one case, it might have been ... I don't know. That detail I don't know. It was, from beginning to end, completion of a fairly well-defined project.

Hemant Elhence: Before you move on from this, this is such good data. Another question someone asked is on the quality of the work, if 0.7 may have good quality.

Todd Little: That's a great question. Actually, the interesting thing again, is that the team that was the lowest estimate and the best actual compared to their estimate, was also the highest quality. The one that produced the simplest solution and the best answer had the highest quality and the ones that were producing the larger systems that took longer to do it, actually had worse quality. Yes, he had comparative quality data in his analysis as well and some costs of ongoing maintenance estimates as well from it.

Hemant Elhence: Interesting. Okay, thanks.

Todd Little: It's a great paper. I don't know if this has actually been published yet. I was a reviewer on it last year. It is out on his website. It's available. If anyone wants it and can't find it, send me a note and I can send the paper. It's a fascinating paper.

Hemant Elhence: Excellent, thanks.

Todd Little: With that one, given the ranges that we're actually seeing, this one I'm declaring to be busted. This is a challenge that is much bigger than we understand and very few people actually realize how much of a challenge we face with estimation.

Let's look at one of the next items we can see, which is the estimation accuracy significantly improves as the project progresses. We ask, how does estimation accuracy improve over time? We have this item called the Cone of Uncertainty. Barry Boehm first presented this in 1981 showing that, from his view, what he'd see with projects he'd run and

data that he looked at, that basically, uncertainty improved over time based on the different phases of development that the project was going through.

It's an interesting concept. It looks like it might be an accurate thing. In looking at it and actually going back into details, it turns out that Boehm presented this as a subjective view of how he thought the data was. He didn't actually have empirical data to support the view, but he thought it matched about what he'd seen under the data. When he actually did present some data, there was data from 7 projects that were done at the University of Southern California graduate school and 5 projects which were proposals to the United States Air Force, so a total of 12 projects, and each of those had one estimation point that they looked at. To the extent that he empirically validated, that was the empirical validation of this.

I did want to highlight one thing, which is this 4X ratio that we see fairly frequently. This is consistent with what Boehm had been seeing from his anecdotal data that he observed, is that in that early phases, somewhere between, some are 4X, 0.2 vs. 2 on the low bound and a few on the high bound, was 4X is a common behavior that you see.

I searched that since I had the data to actually look and see, do we see this type of behavior that's postulated by the cone? I looked at the Landmark data -- Landmark's actually the name of the company that I was with that gave us the data -- and looked at the estimation error over time. On the left hand side, I'm looking at the actual duration over the estimated duration and the prime axis as well as time. At the end, 1.0 is a relative time. All projects that ship will, by definition, converge to one at that point.

We see a couple of things. One is we have something that sort of looks cone-ish. What we show here is that it's definitely biased to the upper portion of the curve. Consistent with our data that we showed earlier, that we were optimistic but that's not optimism bias here, where the actual is almost always greater than the estimate that we put forward at the time. It does have a little bit of a cone shape. Maybe there's something to it.

I looked at this, but is uncertainty really reduced, based on this? You get this quote from Ibsen: Take away an ordinary person's illusions and you take away happiness at the same time. Are we really reducing uncertainty? The real business question that we're asking is not how much total work we have… We're asking how much work do we have left

and when do we ship? When Dorothy gets the time clock, she doesn't ask how many grains of sand there are in the overall hourglass. She asks how much time do I have left? That's the real question: how much time do I have left?

If we look at remaining uncertainties, the uncertainties of the remainder of the work, the forecast when I'm halfway through the project, I'm not projecting out the total amount of time I have. I'm projecting out just the amount I have remaining. We get a very different picture. In fact, this is not converging necessarily. It doesn't have to converge because if I'm on the last day, and I estimate I'll be done and then actually find a bug and it takes me a week to fix that bug, I'm off by a factor of 5. You don't necessarily expect it to converge at that point.

We're looking at remaining uncertainty and it turns out that we're seeing something very close to the 4X line. That looks like one of the charts distorted a little bit. Sorry about that. Anyway, the line here, the place where we're hitting is close to the 4X level. It's fairly consistent throughout time that the amount of estimation error that we're seeing in the remainder is pretty much 4X, if we're looking at the range between- If we want to have an 80% profit.

There's a good reason why this has been the phase, why remaining uncertainty doesn't change. If we look and we've got a set of stories that we've estimated, we have some uncertainty associated with each of the stories. When we finish a story, we've removed the uncertainty in that story or those stories that are completed. The reality is that there's very little uncertainty that has been reduced in the remaining stories. That uncertainty still remains. The relationship of one story to the other is not such that it necessarily inherently reduces this uncertainty. As that continues, we're still stuck with this remaining uncertainty until we're finally done.

Hemant Elhence: Are you, along with this, busting the myth that there's a learning value that in each iteration of Agile, you learn enough about the technology or the requirement of the domain that you are now able to better estimate the next sprint?

Todd Little: I believe that it's possible that you can get that. What I'm saying from the data that I've shown, that I'm seeing, that it doesn't happen implicitly. It doesn't happen as frequently as we think. I wouldn't count on that miracle happening. Now, there are things you can do to proactively learn about and remove uncertainty. Sometimes that removal of uncertainty is

a good thing. Sometimes that removal of uncertainty actually may be a bad thing in terms of decreasing your ability to respond to change.

I think the fundamental from this and what people have taken away, interpretations of the Cone of Uncertainty, I think much of it was based on a reinforced misinterpretation of the cone, which has been very circular logic and when you actually take data ... I have this data I've also been working with a professor in Amsterdam that has collected over a thousand European projects and validated much of the same conclusions that I had seen. What he had seen is that a lot of it depends on organizational behaviors and some organizations, the uncertainty actually gets worse as the project progresses, and there are some cases where the uncertainty does get better, particularly that happens as a result of proactive design to reduce uncertainties going forward.

In general, what we found, it does not decrease drastically. There's still significant uncertainty. I'll go through a couple more examples of where some of those challenges are.

Hemant Elhence: Sounds good. Thanks.

Todd Little: This one, I think the key word is that does accuracy significantly improve? Generally, that is not the case. It does not significantly improve. The uncertainty is still there and there are things you can do about it.

I'm going to go into the next one. Estimations are frequently impacted by biases and the biases can be significant.

The first one I want to point to is the optimism bias, which we've seen and seen time and time again. In our industry, we are inherently optimists. We think we're going to be able to get things better, at least easier than they are. Our reality is that we have challenges and those challenges, we don't account for those, so definitely an optimism bias.

This version of bias is a little different, I'm going to run through. This is, again, work from Jorgensen, published by IEEE Software 2008. He took the exact same spec and went to four different groups of estimators. The control group is Group D. He gave them no additional guidance other than what was in the spec. They came back with their assessment of 160 story points. He told each of the other ... Group A, he told the guidance of 800 stories. He said, here's a spec. We think it's around 800 story points. In Group B, here's a spec. We think it's around 40 story points, and Group C, here's the spec. We think it's around 4 story points.

What sort of results do we think we're going to get from this? Frequently when I ask this, I get the response, well, everyone's going to give back a response greater than what they were told. Certainly that's one of the possibilities. What really happens here is that people adjust to and are skewed by the number that they were given. The control is 160. Those that were told 800 say, no, that's not really possible. It's not really 800 but we're skewed by that number and we're going to do things somewhere in the 300. 40 came in at 100 and the 4, they said no way that's going to improve theirs up to 60, but you can see, definitely lower than the one that was given the 40 guidance. Just bias and guidance can definitely impact estimation.

Similar thing they asked in one case, was the same exact spec. The only difference in the wording was, they said this was a minor extension vs. new functionality vs. extension. The control group was 50. The minor extension came back left. New functionality came back twice that of the minor extension. Exact same specifications and the estimates were quite different.

Then another group was told future work was going to be at stake relative to control. Definitely they were told to make things much more, try to do something much more efficient in the estimate, but the exact same steps.

Hemant Elhence: One question on the data, Todd. Regarding story points…was there some sort of a consistency across themes using story points?

Todd Little: I think these actually may have been work hours, I don't know. I used story points because I don't know if this is hours or days or what. This was actually consisted unit.

Hemant Elhence: Okay, I'm sorry. Thanks.

Todd Little: Sorry for the confusion there. Bias is definitely a problem. What gets us into trouble is not what we don't know, it's what we know for sure. These are challenges of estimation all around so yeah, bias is significant. Yes, I think this is one of those that we can definitely confirm.

Next one, this is fairly popular in the Agile community. We're pretty good at estimating things relatively. A couple of interesting things. Look at that picture and think about it for a while. We're going to go into some of the challenges with estimation. This is a challenge of estimation in general. It's definitely one that also will definitely impact relative estimation, and that's something called Anchoring.

Anchoring, to a certain extent what we saw just previously with giving the guidance, and given that guidance, we impacted things. This is what Tversky, who was a researcher and was the first one who really pointed out how much of a challenge estimation is, and this is not a software. This is in general just the human mind and how it works with coming up with estimates.

They were doing a bit of a situation where they were, something like the Price is Right where they're getting people to estimate the cost of something. The experiment that they ran is they first had them write down the last three digits of their social security number. There's a very strong correlation between the social security number that they wrote down and the amount that they estimated the item to be, something you would not expect there to be a correlation to, just the fact that those people with high social security numbers would write down a much higher price for the item. This is something called Anchoring. We get anchored to something we had previously and when we look to that, which influences things going forward.

In relative estimation, one of the challenges that we definitely have is relative anchoring. That A relative to B is not symmetric with B relative to A. Again, some work by Jorgensen on this, published last year. He did a lot of this with software estimation and some of the challenges with relative estimation in software. He used a number of examples also from general knowledge.

This is one of them. If we took Austria's population and estimated it relative to Hungary, he came back with Austria was 70% of Hungary's population. Vice versa, when you estimated Hungary's population relative to Austria, it was also 80% of Austria's so neither one was bigger than the other. They were both smaller than the one they were compared to. It's not symmetric, which is one of the huge problems with relative estimation. It's a problem with estimation in general, is what we're comparing to in anchoring challenges.

The other problem I wanted to look at was just looking at this is the element of dimensionality. This picture here, I took from somebody. It's a financial trainer's picture that sat on the website. If you look at it, the big rock looks about twice as big as the small rock. The challenge here is what's that looking at? The human mind tends to overemphasize one dimension, one dimension being vertical over horizontal.

If I'm looking at the size of a rock, I'm not just looking at one size. I'm looking at the full size of that, which is I'm looking at the volume. If I look

at this from a perspective, is that 2X as big? It's only 2X as tall but with rocks, we have three dimensions, which means it's actually 8X as big. Again, we're lulled by 4X, which is consistent with some of what we're doing in software as well.

The main thing is this is a multi-dimensional problem. Relative estimation is, when looking potentially at one dimension, it's not necessarily any better at looking at multiple dimensions. It's not that there's an inherent problem with relative estimation. It's just that we're not any better at relative estimation than any other. It's not a miracle that relative estimation is better than doing some other approach to estimation or absolute estimation. We suck in relatively different ways.

Are we pretty good at estimating things relatively? I think not. We're not any better, but there are reasons I still use relative estimation. I think there are good reasons for using relative estimation, but it's not this reason that we're pretty good at it. It really comes to the next one, which is that velocity and throughput is a good tool for adjusting estimates.

This one is pretty much correct. What we do is if I look at, in particular, like a burn-up chart, because a burn-up chart shows me my velocity as relative to my scope creep velocity. I can look at that and that helps me understand my projection of where I'm headed. As an overall aggregate, it's a nice convergence approach.

Velocity is taking ten. It's using that relative estimation. Its converging story points into a time via iteration. If I take a total number of story points I have remaining and I know my net velocity, which is my actual velocity less my script rate and I can get some indication of how many story points am I burning down per iteration. I can then turn that into the number of iterations I have remaining and over time, I can convert that to time and I can project out where I'm at.

Relative estimation works here because we're in story point and our velocity is also in story points. By having the same units and having consistency in the approach, we can convert that into a projection time. Velocity is a good tool, but it's not a silver bullet. It's great for dealing with the bias in particular. It doesn't really do anything to address the uncertainty. Velocity adjusts things. It doesn't necessarily remove my remaining uncertainty. It's a good tool for adjusting estimates, definitely to remove bias, not necessarily a cure-all to account for overall uncertainty.

Let's go into the next one: We're a bit behind but we'll make it up in testing since most of our uncertainty was in the features. This is a great article but Lan Cao called “Estimating Software Project Effort: An Empirical Study”. I like it for three reasons. First, she references my work so that's great. Second is, she more or less confirms, has results similar to my work. The third, which is the real reason I like it, is because she takes a look at the differences. She has task estimations for both the features of a project as well as the defects in the project. While she doesn't come out directly and say it, she presents enough data that you can infer this from the data and that is the uncertainty in the defect is twice the range of the uncertainty in the features.

This is, I think, a very important finding. There's a good reason for this. When we're trying to understand how long it's going to take to fix a bug, we often don't even know what's causing that bug. The cost of the time to de-bug and actually, the actual time to fix it may be trivial, but the time to find it is enormous. Whereas with features, we have to have a pretty bounded understanding of what that is in the meantime.

The challenge here, and this is one of the main reasons of why you want to address the defects early is because if you're addressing your defects late, you're pushing a huge amount of uncertainty to the end, which is the last place you want your uncertainty. You want to get your uncertainty out of the way early, get those bugs under control and that's the means by which you'll be able to actually have control and predictability in your project. As we know, we're not going to be able to make it up in testing. It just never happens.

Number seven, scope creep is a main source of estimation error. This is a fairly consistent problem the industry has faced. You actually can do things to control scope creep. Actually, we in the Agile community like to look and say well, actually some scope creep and some new information is welcome and we want to take advantage of that because we want to be able to adapt to change better than our competitors.

Scope creep is a reality. Things are going to change and we're going to learn by Jones who had some great metrics on this, a little simplified here. 2% per month of your scope is what you have happening on a change per month or 27% per year.

It doesn't sound like a lot, the 2% per month, but actually if you're looking at this chart here is what- If you had totally ignored scope creep and you had a project ... If you had a project and on the x-axis here, we have a planned duration and the initial plan. If we hadn't accounted for

that scope creep going in, there's the ratio of actual to estimate over on the y-axis. If you get out there into the 40-month project, you're looking at it taking 5X that because the scope creep is eating away at everything that you're making progress on from a velocity perspective is getting eaten up by scope creep. You're just treading water and if you go out much longer than that, you never get done.

Actually, the inverse plot of this is some of the data that's published by Standish Group and reported by Craig Larman, book on “Iterative Development: A Manager's Guide”. This is project success as defined by Standish, which is on-time delivery with all features and on budget. But you see this is that success as a function of project duration as you get down into the 36-month time, the chance goes down to zero. That's what we like to do in the Agile world is keep projects as short as possible so that we actually have a time horizon that we can work with. It's the means by which we deal with the inherent uncertainty in place. Scope creep being a major source of estimation error, I would give that one a confirmed.

Number eight is having more estimators, even if they're not experts, improve estimation accuracy. This is an exercise I do when I do this session interactively. I have people at the beginning of the session individually estimate the number of jellybeans in the jar. I then have them do a group activity where they do it together as a group and we also then look at the overall average of the individual estimates and see how that works.

I used this as an example for a couple reasons. One is to show how huge the estimation ranges are in this or for the individual estimates, this is on a normal line. One would be a perfect estimate of what's in the jar. What I find is that fairly consistently, I've run this 5 or 6 times, the range I get is from about 20% of the total up to about 3X the total, for a range of 15X between the low and the high. Fairly consistently in that ballpark.

When we have the groups do the estimate that goes drastically down to a range of about 0.75-1.5. We cut the range of the uncertainty down by 2X. If I just take the average of the individuals and then look across each of the different groups I've done, that tended to range from between 0.8-1.2.

This is very repeatable. It's been mentioned in James Surowiecki's book “The Wisdom of Crowds”, which we'll get to now. This is an interesting perspective of the “The Wisdom of Crowds”. Of course, it's gotten a lot of press over time, crowdsourcing and things like that.

I had read earlier, a book called “The Extraordinary Popular Delusions and The Madness of Crowds”, which takes an opposite view, which is when crowds go mad. Sometimes crowds are smart and sometimes crowds are mad. If we look from the wise crowd, certainly the jellybean experiment shows that. You can also look at from the Who Wants to be a Millionaire program, the audience was correct 91% of the time, way higher than the so-called expert phone-a-friend. That was barely 50% correct. The audience was almost always smarter than the so-called smartest individual.

On the other side, in the Madness of Crowds, one of the big stories in this book is the Dutch tulip mania in 1637. This book, The Popular Delusions was published in mid-1800s. In referring back to this Dutch tulip mania where a single tulip bulb was worth of an order of the equivalent of a million dollars today. Everyone was frantically after the tulip bulbs and then it crashed. I guess, though, looking at our stock market, we're having similar challenges and a huge amount of uncertainty these days.

The thing is, when are the crowds smart and when are crowds not so smart? When are crowds mad? It really comes down to diversity. When crowds are diverse enough, they are very smart. When crowds are not diverse, when they have a very similar mindset, that's when crowds go mad. That's why it's important in estimation to make sure we're bringing in multiple, diverse viewpoints, so it's not just developers. Its developers and testers and product owners as well, the entire team.

One of the things I'll do is ask the team, when do they think, they're going to be done. I ask it anonymously. That way, they can give a feedback without fear. This is the distribution of the debates. On the x-axis are the debates, and the y-axis is the percent of people that are falling into that category. We get a distribution that shows us roughly where are people at. Mostly, probably developers were in that first bunch to the far left. The testers are the next two people that are a little bit more bashful about when we're going to be done.

It gives some good background of this team. I think this team did make about the average of what this was so it was two things. One is you get a good balance of the understanding from the team. The other is you have a conversation around it and open up the possibility to explain why they're concerned about it and what the overall team might be able to do about it.

Having more estimators, even if they're not experts, improves estimation accuracy. This is one of those things that's generally confirmed.

Let's move to nine: Project success is determined by on-time delivery. Certainly, this is the definition Standish uses with their fairly frequently quoted successful challenged and failed projects. The real question is why do we care about on-time delivery? Why is time such a big thing for us? What I really try to get teams to look at is what we call the cost of delay. It's a term that's come out of the community, cost of delay. What is the situation we're in with our product delivery and how much does it cost us if we're delayed?

This that we're looking at on the y-axis, the relative value, the value lost as a function of time of delivery, and there's some optimal time that we're targeting to deliver, but then we're looking at are we in a mature market with a strong barrier to entry where being late is not really a competitive threat? Are we in a situation where we have a highly competitive rapid market change and it's really critical that we get something to market or are we in a situation where we have a hard deadline like the Olympics where we've got to deliver something for the Olympics and if we miss it, we're waiting four years until the next one?

We also have to make sure with this cost of delay, we're not having the wrong priorities. This is a story from the Ford Taurus. First, we have the Ford Taurus in 1986. The Ford Taurus was a hugely successful release in the first round in 1986. It saved Ford Motor Company. All the American automotive companies were in serious trouble in that time. Chrysler had just gone through bankruptcy. Ford came through. Taurus was hugely successful. Project manager was going to make sure this was the ideal car. He had lots of focal groups, lots of prototyping, and lots of things to just make sure that everything he was doing was focused on delivering the best consumer experience possible with this automobile.

Problem was, he was 6 months late. His reward for delivering and saving the company was to be fired or demoted. I'm not actually sure of the storyline here. They're both stories about whether he's fired or demoted. In any event, it wasn't a pretty picture for the project manager, the guy who had just saved the whole company.

Hemant Elhence: Time check. We're just 5 minutes away from the scheduled finish and I know you've got more content so just, if you could speed up a bit.

Todd Little: Yeah, we're good. Anyway, the second rep of the Ford Taurus, what do you think the project manager made sure? He made sure he was on-time. He cut all the corners he could. He did none of the focal groups. Terrible, terrible consumer reaction to the second rep of the Ford Taurus. It's the wrong priority. We want to look at the cost of delay but we also want to

look at the cost of crap. We don't want to be delivering crap on-time. I've seen this happen too often in the software industry where we're just so focused on delivery, we end up delivering the wrong thing.

When we look at an on-time metric, is on-time what really matters? Again, this is a plot from my paper. I lifted Lanmark's hundred plots and what we had. I also looked at this little project, Microsoft Windows WinWord 1.0. WinWord 1.0 is oftentimes used as an example of a disaster project because the initial schedule was one year. It took them five years to deliver. I look at WinWord 1.0 and I think, what happened? What sort of value did WinWord generate?

The reality is that WinWord made a huge change in the dynamics of the word processing market, and in fact, eventually, totally destroyed Word Perfect, which had a 50% market share. Eventually Windows took over and absolutely dominated this marketplace. Huge amount of value generated from this product, so delivering what actually matters is key, far more than delivering something that was on-time. This is a challenge we've had that was- Hubbard refers to this as measurement inversion. We tend to measure the things ... We often have this inversion where we measure the lowest value items the most frequently -like cost and time, and the things that are highest value like value deliveries are least measured.

Project success is determined by on-time delivery, too often not true. I declare that's busted.

Then we go to the last one here, which is elimination is waste. The real business questions we have, is this worth doing? What is the priority? When is the target time to ship? What is the critical scope? Do we have the right investment? These questions around what is the cost of delay? These are the business questions, the reason why we do estimation. Too often, we're not doing estimation for that reason. We're doing estimation for other reasons, to feel the illusion of control or something else.

When we're doing estimation for the wrong reason, it's plausible but we're actually wasting time. Every time we spend on estimation is time not spent on delivery. We need to understand how much value we are gaining from that estimation, because we may be fooling ourselves into an illusion of control that we don't really have.

Just a quick wrap up. Now what do we do about it? I like to have teams use a simple model like this to estimate. To get my priority, I look at value

and cost and have estimates across both of them, do just enough estimation to help identify the priority and understand where we're at.

I like to use this thing called and A/B/C list. It's similar to MoSCoW, the must's, should's and could's, but I use different wording because I don't the implications of should. Should sounds like if I don't deliver, that's bad. I like to switch it around to A/B/C. I used to say A/B/C was MoSCoW for pre-schoolers because it's simpler. Also, I think there's a fundamental difference in how I implement it. A's must be completed and I'll shift the schedule if necessary. The B's are wishes. I'm not expecting to get that many of them. I'm getting some of them, but they're wishes. It's like a Christmas wish list. I expect some things but I'm not expecting to get everything. Then C's are not targeted.

If I have more than 50% of the planned effort allocated to A's, then the project's going to be at risk. I also make sure that the size of the scope creep, this is just an example. 500 story points to start with. My velocity's 25 points per 2-week iterations, my 2% per month gives me 5 points but it's going to cost me, which means my net planned velocity will be 20 points per iteration. Even though it's only 2% per month, my net impact on my velocity is 20%.

What this does when I look at my timeline, if I have my backlog lined up and 50% of my time allocated to A items and B items theoretically the next 50%, what I actually deliver, assuming things go well, I deliver all my A's, but I'm going to have some change. Some of my B's I won't get. Some of my C's will become more important. Some D's that I didn't even think about will start showing up.

The other thing that can happen is that I happen to hit bad luck with my A's. It could end up taking up my entire delivery time. I provide that buffer through this approach and I'm able to have some degree of control over a target date, have some variability in scope, and have conversations consistently with business team to figure out what's the right approach, should we be adjusting schedules, should we be adjusting scope, and have that conversation on a regular basis.

The attracting metrics, the burn-up chart, continue to monitor the quality because one of the best ways to cheat on a burn-up chart is to allow quality to suffer. If you let quality suffer, you're just postponing problems until later. I like to make sure that the defect log is close to zero and that I'm doing all the right things and monitoring quality as well. I ask the team to have conversations around the cost of delay.

With that, let's wrap up with my contact information. Hemant, you want to take over?

Hemant Elhence: This is good. I think this is excellent content so I didn't want to lose any of this. We have come to the end of the hour, a minute or so over, so I think we will just take questions offline. If you guys have any questions, feel free to ping us. We'll have the questions in our data anyway. We'll follow up with you if any of those require follow up. Thank you so much, Todd, for covering this content. I wish we had a little bit more time. There's so much valuable, this topic of estimation. Stay tuned, all of you, for next month's seminar, which you'll hear soon about. Thanks, Todd, again.

Todd Little: Okay, thank you.