Hanselminutes is a weekly audio talk show with noted web developer and technologist Scott Hanselman and hosted by Carl Franklin. Scott discusses utilities and tools, gives practical how-to advice, and discusses ASP.NET or Windows issues and workarounds.

Text transcript of show #119

June 26, 2008

What is Done? - A Conversation with Scrum Co-Creator Ken Schwaber

Scott chats with Ken Schwaber, the co-creator of Scrum, agile advocate and a founder of the Agile Alliance. Scott asks 'What is the definition of Done?' and gets a more complicated (and more interesting!) answer than he bargained for.

(Transcription services provided by PWOP Productions)

Our Sponsors

Lawrence Ryan: From hanselminutes.com, it's Hanselminutes, a weekly discussion with web developer and technologist, Scott Hanselman, hosted by Carl Franklin. This is Lawrence Ryan, announcing show #119, recorded live Wednesday, June 18, 2008. Support for Hanselminutes is provided by Telerik RadControls, the most comprehensive suite of components for Windows Forms and ASP.NET web applications, online at www.telerik.com, and by the CodeBetter Blog Network, delivering tried and true solutions to real world problems for building better software, online at codebetter.com. Support is also provided by .NET Developers Journal, the world's leading .NET developer magazine, online at www.sys-con.com. In this episode, Scott talks with Scrum creator and founder of the Agile Alliance, Ken Schwaber.

Scott Hanselman: Hi, this is Scott Hanselman and this is another episode of Hanselminutes, and I'm sitting here in Oslo, Norway, with Ken Schwaber co-inventor of Scrum, founder of Agile Alliance and all around agile advocate. We were lucky enough to be on an agile panel this morning with a number of agile luminaries but I'm sitting down here with Ken and we're going to talk about agile, agility software process, but you had said sir that you thought that one of the things that needed to be talked about was 'What is Done?'.

Ken Schwaber: Yes.

Scott Hanselman: Which was surprising to me that that was what you thought was the most interesting thing we should talk about, 'what is the definition of done'?

Ken Schwaber: Well, done is a very simple word and Bill Clinton started this with simple words when he talked about this during his presidency. Done is something where if I come in to a company and they say, "Oh, we're using Scrum." It's a smell for me about whether they're using Scrum or not. So, if I turn to the person who is a customer or even a team and I say, "What is your definition of done?" and they give me kind of a blank look, to me this is a smell that they haven't addressed one of the key issues in Scrum and that is when they select some items that they are going to do for a sprint that is some items, some requirements and product, they are clogged items and storage that they say they will do for the customer during an iteration which we call a sprint, and they don't know what Done or Do is, this certainly raises a question to me about how they know what they're doing because if they don't know what Done is, that is, is it coded, is it unit tested after it has been coded, is it maybe even refactored, does it include a design review, a code review, does it include performance testing, what does it include? If that person or their team that's working on things for the customer doesn't know what Done is, then you have to ask a question about how do they know how many things they should select during an iteration. If it's just coded and maybe unit tested, maybe they can do 10 or 12 or maybe even 20 items during a whole sprint. So if it's both analyzed, designed, there's been unit tested, there's been coding, there's been testing, there's been refactoring, and a whole set of things are needed to be done for it to be potentially shippable, then maybe they could only do one thing. So you have to wonder how they know how many things to select from the product backlog to do if they don't know what Done is. This also raises the question of what sort of expertise or engineering skills do they have if they are not sure what they do when they select something to be done and this, of course, then raises the question of undone work. When they get done with the sprint and they've done five things for a product owner, the product owner, if maybe the total of done things that they need are 30 might believe that they are 1/6th of the way complete and ready for shipment. However, if the team's definition of Done is only coded and unit tested, maybe the product owner is only 1/40th of the weight done, and when the team gets done with the 6th sprint and says, "Here we are product owner, we're done," and the product owner says, "Great. Let's ship." Then the team has to look him in the eye and say, "Well, that's not quite the definition of Done we had. Now, we've got to do all the rest of the testing and refactoring and the stabilizing of the code to make it so it is really shippable so maybe in about two or three months." So, this is why Done for me tends to be a pretty big issue of engineering skills, of the relationship between the product owner and the customer, beyond engineering competence of the team that is doing the work.

Scott Hanselman: It sounds like, from the way that you're painting it, that Done is the only point. It also seems like one could build up a great deal of technical death even within just a short iterations but more importantly that sitting down in its step zero saying what is done for us could expose a number of technical deficiencies in an organization. You're illustrating that there could be cancer in the organization like we have virtually no technical writing or we have insufficient integration testing so we might not even be able to appropriately begin a sprint until we work these other things out.

Ken Schwaber: If I listed a number of things to be done to a customer requirement before it's potentially shippable, I've tried that and the list is about 45 to 50 items, things that need to be done, and if a team is only capable, if you go through this with a set of group of engineers, group of developers which includes programmers, analysts, designers, QA people, documentation people and say how much of this can you do in a sprint, it's very often that they can only do 15 or 20. They're going to look at a whole bunch of those like the performance testing, the quality testing, the regression testing, then they're going to say, "Well, we don't know how to do that yet

in a sprint," and to me that remains undone work. If I do six sprints, at the end of the first sprint, I have some undone work, at the end of the second sprint, I have some undone work, and it keeps accumulating. Unfortunately, it doesn't accumulate linearly. As you don't do work sprint after sprint, it piles on so you're not doing some work that's piling up to what you didn't do in the first sprint, third sprint, second sprint, and so the amount of work that you're not doing tends to accumulate in some of logrhythmic pattern, not sure exactly what it is but it is certainly more than linear.

Scott Hanselman: Interesting. It feels like when I did my first Scrum, when the CTO of the company that I worked at before I went to work for Microsoft, brought in some Scrum training and we had a number of folks becomes Scrum masters. They put the entire division, every developer went through Scrum training and we said we are doing Scrum as a company. We dedicated ourselves to it and we sat down and we started to ask ourselves what was done, sort of realize how little we have been doing in the past, which was interesting, but more interestingly, how much work needs to be done made us feel like we were getting less done. I guess what I'm trying to say is that we were concerned when looking at Scrum that what we perceived as the most important piece of work, the code, is getting squeezed and we're going to spend all this time writing documentation and doing testing things like that. Is that a common perception that someone says that "Gosh, the speed at which we are getting things done is less because look, we are having to do all this administrivia."

Ken Schwaber: Yes.

Scott Hanselman: Just to make someone feel like we are done.

Ken Schwaber: I believe in every programmer's head, there's a little person who has a metronome that goes at a certain pace and if we don't go at that pace, then we're falling behind.

Scott Hanselman: Right, I agree.

Ken Schwaber: And so if we take the time to actually test our code and this is a whole range of testing it and perhaps even document it to make sure that it works as documented; then we are not going as fast as we should. At that point, we kind of panic and we throw all that stuff over the board and say we'll do that later because that's not really customer stuff, that's other stuff.

Scott Hanselman: Right, administration. It's just administrivia we call it.

Ken Schwaber: Right. Well, acceptance test driven development blows all this out the water because you're defining the acceptance test that will prove whether it works right at the front and then you go off and you document it. You set up the test for it, you set up the code for it and then you try to see whether it works as the test described and it works as the documentation describes, and that's a complete piece of code. If it is not, you circle back and get it complete. If it is, then you go back and pick up another piece of code. Now, that's talking about everything that's customer phasing being done within a piece of word called Done.

Scott Hanselman: It seems to me like programmers and engineers are deeply interested in delivering code and if we're not slapping the keyboard and making our curly braces or whatever language we're using, we're not productive. That means that we are fundamentally disconnecting with the whole point which is to make a happy customer.

Ken Schwaber: Well, this goes to the point of a flaw, almost a genetic flaw that has occurred in programmers that's come up from waterfall and this is that we are willing to cut quality to increase velocity or to increase the drumbeat that the little person has in our head.

Scott Hanselman: Exactly.

Ken Schwaber: And so I can sit at a desk and I can do things the right way, maybe think about how something is supposed to be done and where to fit the design, and what after we refactored to do and then the coding for the test and that might give me an hour, but if someone tells me that this is really important, I've got a deadline to meet and stop fooling around, this is really serious, I can do that same work in 10 minutes by only doing the coding and slapping it onto something written and refactoring it. So that's where we come up with the idea that only the code is what matters.

Scott Hanselman: Yeah.

Ken Schwaber: Unfortunately, with this result in them, because we've cut the definition of Done, it's a Done deficit or undone, which is where we strangely enough get hard things like high maintenance, low sustainability software.

Scott Hanselman: It feels like there is a synergy between the Scrum methodology and what has been happening in productivity with David Allen's Getting Things Done way of thinking. It's kind of like the new Stephen Covey. I'm not sure if you're familiar with David Allen's Getting Things Done.

Ken Schwaber: No.

Scott Hanselman: It's called GTD and it has been -- the whole concept of his productivity style is that one has a mental backlog of all the things that are causing new psychic weight, that's what he calls it, and psychic weight can be anything in your life from

write some code to clean out the garage and you can't release that psychic weight until you actually had it written down and that's backlog items, put every single thing that needs to be done. Then you can release that because it is in a trusted source, and then the most important thing about his solution which I think can be applied to something like Scrum is what is the next action required to move this thing forward, not at the project level, but what is the very next action. For example, if I'm going to paint my house...

Ken Schwaber: Yes.

Scott Hanselman: That is my backlog item, paint my house. What is the very next action required? Well, it might be something very trivial like get the phone number of the painter.

Ken Schwaber: But it's what I know, what it has to be, rather than me looking for any painter I can find under the table and starting to paint.

Scott Hanselman: Exactly, but the programmer's mind wants to just start painting, "I've got some paint. I'll just start slapping paint on the wall. It will work itself out."

Ken Schwaber: Well, the most suspicious thing in any organization are teams that have a velocity of 10, 12, that might be the average within your organization, but another team is able to get a velocity of 20, 25. Amazingly, if you look at their code and the tests around it, they have changed the definition of Done to increase that.

Scott Hanselman: Say something about velocity because I don't think necessarily, all of our listeners are familiar with the concept and the numbers associated with Velocity.

Ken Schwaber: If we have a list of things that a customer wants to have done, this is how many of those things we're able to get done as a team over a period of time which might be like a monthly iteration. So, velocity might be 20 pieces of requirements or a product backlog that we're going to get done over a sprint. Now, if we are able to change the definition of Done to minimize testing, to minimize design, to minimize refactoring, you can raise that incredibly. However, if within an organization, everyone has the same definition of Done and it's something that's potentially shippable, then I can really compare one team to another.

Scott Hanselman: Interesting. I think that that idea of having a common agreement across either groups or divisions make the statistics have value. Otherwise they're useless, they're just numbers.

Ken Schwaber: Otherwise, they are useless and an interesting way of making so the statistics are relevant is all the teams that are working on a similar set of software, let's say a similar release of software, have to have the same definition of Done otherwise their code won't integrate during the sprint and won't be integrated at the end of the iteration or sprint, and so this becomes an organization why definition rather than just simply for a single team.

Scott Hanselman: Let's just take a moment right now to thank our sponsors and we will come right back with Ken Schwaber and we'll talk a little bit more about what is Done.

Do you know how to make the possible out of the impossible? Well, the .NET ninjas at Telerik do. They just released a huge pack of web controls all built on top of ASP.NET AJAX that will help you build impossibly fast and interactive applications in no time at all. They have made the impossible possible in desktop development. If you think you can't have a Carousel component in Win Forms, well, you can. Their Windows Forms Suite features a super powerful GridView control and 32 other crazy desktop components that will give you dazzling WPF-like features, but in Win Forms. They do the same thing in reporting solutions with a new design surface like nothing else. It looks just like graph paper because of the advanced page layout capabilities. It makes it feel more like a graphic design software than a reporting solution. Go check them out at telerik.com and be a .NET ninja. Thanks for listening.

This week's Hanselminutes is brought to you by CodeBetter.com. The CodeBetter.com Blog Network is made up of over 20 industry leaders and speakers who are passionate about delivering tried and true solutions to real world problems for building better software. These guys are not only our sponsor this week but they are also my friends. The CodeBetter.com Blog Network, it's where industry leaders blog. You can find them at CodeBetter.com as well as Devlicio.us.

And we're back. We're talking to Ken Schwaber about what is Done. You were going to say something sir about the relationship that happens between the customer and the developer and how the definition of Done can affect that relationship.

Ken Schwaber: Yes, we've talked about how Done is necessary so a team knows how many items to select and so it knows what development process goes through within a sprint to create something and so that there is no undone work left at the end of the project for the team to stabilize. However, this changes a long-term, a long time-honored relationship between customers who we now call product owners, and the developers on Scrum team. Normally, a product owner would come in if they needed something more done than they had anticipated in a project and they will tell the team, "Hey, we've got some more things. We've got some more stuff for our customers. We've got some more functionality that

we need to get into this release and it's just critical." And the team would look up and say, "Well, yeah, but we don't have any more time to do that," and the product owner will look at them even more clearly in the face and say, "You don't understand. This is really, really important." The team would say, "Oh, now we understand. It's that important that if we don't do it, you're going to go to our managers' managers' managers and we're all in deep trouble if we don't do it." So at this point in time, the developing team does what would do, what it always has done in this relationship which is it cuts quality, that is it reduces the amount of stuff that is included in what's done to simply increase the amount of stuff that can be claimed as done, and as I have mentioned earlier, they can drastically increase their velocity by doing this. They might increase it by up to four, five, certainly enough to accommodate almost any increase in things that a customer's product only has to be four. I think this has led to a deep seated belief by all the customers that in general, we sit around as developers with our feet up on the desk. Maybe I'm playing with surfing the web or looking at pornography or doing something but it's certainly not doing the work that they want, and all they have to do is tell us that it is really important for us to do it and we will put our feet down and we will sit forward and we will really crank out the code like we always have. What they don't know is that we cut quality by reducing the definition of Done to Do it. So this changes the relationship between the product owner and the team. The team now has the definition of Done. The definition of Done is something that is potentially shippable. This is something which is poor quality, it's maintainable, sustainable, and enhance-able code, and yet the product owner doesn't necessarily know about this and so they will come to the team and say, "Guys, you got to get more done." And the team can't. The team is stuck with the definition of Done and so their Velocity is fairly static. You can't change your Velocity by more than 5% to 10% within any sprint by any technique other than cutting quality. Adding people won't do it, increasing your engineering tools won't do it. So they have to really look back at the product owner and say, "Excuse me but this is our definition of Done, we can't do anymore." So with the definition of Done being in place, the team can no longer produce more by reducing the definition of Done. This leaves the product owner with the question of how do they get changes get taken cared of? How do they get the work that they are used to asking the team to do at the last minute done? Scrum gives the product owner, the customer, another variable that they can work with that they've never had before. This is the ability to iteratively, incrementally, build the product. In the PS, they had to say, "This is everything we want. Here it is, do it." Then later, come in and say, "Oh, by the way, and do this more." Now, what the product owner can do is they can ask for the product to be built piece by piece by piece. Highest value piece, then next highest value piece, next value piece, and when they get done with all the functional that they think is valuable, they can stop. So this means that at any point in time, they can trade off functionality that's in cue to be built with other functionality that is more valuable. This works extremely well when you take two statistics into account. One is 35% of all functionality changes during a release. So this is 35% of the cue of work that a product owner had wanted before that's open for change, and 50% to 60% of all functionality in any release that's rarely or never needed. So what we've given our product owners is the ability to piece-by-piece managed what they are getting into the release so that they can constantly optimize the value and return of investment. They will never do this as long as they are used to believing that all they have to do is tell the developers to do more and the team will. They will only do this when they believe that that is not an option and they have to optimize the value of the release rather than just trying for more and more and more. This is a big change for our customers. This is a big change for our developers. Simply by introducing the value of the word Done, we have changed a relationship and we've also started to increase the quality, sustainability, and the team ability of our products.

Scott Hanselman: You talked about the prioritization and deciding that what I need to get done once I've defined what Done is, I can say I want the most high value things first. How do those business needs and ranking the order in which I want to finish my backlog items, how can those constraints be reconciled architecturally to finish this most important business thing that we require these designs or these underlying or I simply can't do priority one thing before priority five because of the way that I see the design in my head. The natural value order of the business need may not necessarily be friendly to the architect.

Ken Schwaber: Right, usually we're used to thinking of architecture as being done before we start developing software and then we hang the software from the architecture and the infrastructure. If we are listing the work that the customer wants done, we're listing the requirements they want, we're also listing the nonfunctional requirements that are needed to support the functional requirements at the PC, at the security, at the other requirements technique. So those are the top priority product backlog items that need to be done. In Scrum, every sprint has to have at least one piece of business functionality, so in the very first sprint, we may have one piece of functionality and a lot of architectural infrastructural items being built. In the second sprint, we might have a little more business functionality and a little less architectural infrastructural work being done. This means that in every sprint, some of the architecture and infrastructure is going in but it emerges, we don't build it up front all at once. We let it emerge based on business requirements. This means that at any point in time, we'll never have more architecture than the

business functionality that we built demands. It also means that the code that we write had better be very clean, very refactored, and very well documented, otherwise, when we get to the second sprint and we try laying more code on top of it, we will have a bloody mess.

Scott Hanselman: I see, so the more appropriately refactored your code is, the more friendly it is the code itself being agile enough to be modified for future requirements.

Ken Schwaber: Exactly, and this is where that word Done really comes into play because if you take a short cut with Scrum, it's going to catch up with you within three or four sprints, not three or four releases.

Scott Hanselman: So what about cross-cutting concerns about things like security throughout and logging throughout and things that don't -- the kinds of backlog items that one finds hanging around, sprint after sprint. Maybe they are poorly written backlog items because they're something like logging.

Ken Schwaber: So you might have logging if it's a product backlog item in the first and it's logging for this one piece of functionality. So you're only putting enough to do that one piece, and in the second sprint, you're doing a logging for another piece, then third piece, or you might hit logging and break it down into customer needs logging and we will be logging at this type of capability. So you decompose it further and further. So you're putting in just enough to support the business requirement.

Scott Hanselman: What about when a requirement takes a dramatic right turn in a direction that maybe you didn't see? If someone builds a logging infrastructure to log text files, suddenly the customer changes the requirement, you need to log into the cloud and we're just simply not prepared for that.

Ken Schwaber: We wouldn't have been prepared if we had tried to include the architecture up front. The only difference is that we want to build a lot of things that we have to tear apart. Instead we will be able to just start from where we are.

Scott Hanselman: I see. I'm hearing you say that it's that accidental complexity that comes from trying to think too much about the architecture that we might need…

Ken Schwaber: Yes.

Scott Hanselman: As opposed to being driven by exactly what we are told we need.

Ken Schwaber: If we look historically, we're driven to build architecture and infrastructure at the start of the project. If you're using waterfall, that is because if you change any requirement at the start of the project, it will only cost you a dollar. Sixty percent of the weight of the project since we build all the architecture and infrastructure at the start, it might cost us a hundred dollars, so we try to be perfect up front. If we are using the ideas, of refactoring in merchant architecture, then we don't need to be perfect. We're constantly adjusting it to meet the requirements. However, this does require tremendously good engineering skills which remove your ability to devolve from the word Done into some crap.

Scott Hanselman: It removes your ability to fake it.

Ken Schwaber: And completely. If you try faking it with enforcements, you'll be caught.

Scott Hanselman: Yeah, interesting.

Ken Schwaber: So suddenly you have a reason to write really good software and you always have a reward for writing really good software.

Scott Hanselman: It's interesting, a number of people had said to me that -- I have said this before -- that agile is using it as an excuse to be sloppy but I've found when I have been working with really good Scrum masters, it's a much more formal process than really any process I've ever worked on.

Ken Schwaber: It has consequences at the end of every sprint if everyone is paying attention. The Scrum master is the one we hold responsible for the product without knowing what's being seen at the end of the sprint. If the definition of done is that something will be completely done and tested, the Scrum master is not allowed to let product see something that does not meet their criteria. So they likely judge the referee in a football field.

Scott Hanselman: That brings up an interesting question as in what does a failed sprint look like and what do we do about it?

Ken Schwaber: Failed sprint is an awful phrase. Sprints don't fail or succeed. Things simply happen within them.

Scott Hanselman: Okay.

Ken Schwaber: So if we select some product backlog items which have lots of architectural work in it and we don't get done with that within a sprint because it turns out bigger than we thought, we will re-estimate the amount of work remaining on those items and put them back in the product backlog, hopefully the product owner will reselect them for the next sprint but they are not done. They are not demonstrable. Simply, there remains less work to be done on them.

Scott Hanselman: Can it be possible that one would have a sprint and maybe because of the length of the sprint or the inappropriate size of the backlog items that they have nothing that they can demonstrate at the end of the sprint?

Ken Schwaber: Yes, absolutely. Wonderful. Then we sit down with them and say "Wow, that was really pretty terrible. So how can we now select a more appropriate amount?" Typically teams will by the time they get into their third sprint know how much they can select and how much they can do within a sprint. It tends to be a self-learning process.

Scott Hanselman: So when a team is decided to start using Scrum, they don't have to necessarily take it all at once. They can just pick something and try to improve that.

Ken Schwaber: Yeah, absolutely and often when they think they're doing it all at once imperfectly, they will discover that they're not and what they do have to improve.

Scott Hanselman: What kind of preparations should they make for management in a sense of we are not going to become immediately and incredibly effective just because we have moved from one process to Scrum? What can I do to prepare, I mean how much overhead is there for the startup of an agile practice at a previously not agile shop?

Ken Schwaber: None. If you prepare, you will be preparing what you think is the most important step to prepare and you have no idea what the teams are really going to need. So, far better to let a team start with some product backlog that they can find somewhere and do their best to turn it into something that is done by the end of the sprint and in the process of doing this, you'll find everything that they don't know, everything they need to know, and those then become the things that you can put in place to help them.

Scott Hanselman: It sounds like the only overhead is just the act of deciding.

Ken Schwaber: Yup.

Scott Hanselman: This is kind of Schrödinger's Cat view of agile where you're not going to know until you just open the box.

Ken Schwaber: Absolutely and this I think translates back to again the waterfall approach of let's plan and get it perfect before we start. First is the Scrum approach which says there is so much complex you can never know what's needed, so just start and then you'll know what you need. Feels far riskier, but in Scrum your risk is never greater than one sprint.

Scott Hanselman: Very cool. Well, that's all the time we have got and I thank you so much for sitting down with me today, Ken Schwaber, and we'll see you again next week on Hanselminutes.

Ken Schwaber: Thank you.