SYNERZIP
Agile Software Product Development Partner

Transcript: Lifecycle of a User Story!
Presented by Mike Hall
Webinar in March 2015

Hemant: This is Hemant Elhence; I'm the moderator for your session today. Welcome to this month's webinar and I'll introduce the topic and the speaker in few minutes..

The topic today is Lifecycle of a User Story, so we'll use the analogy of lifecycle. Today we'll examine and do a deeper dive on the user story technique, how user story is more overtime and during the software project. To cover this topic we have our frequent presenter, Michael Hall with us.

Michael is CEO and Founder of Three Beacons, and he is himself actually a software practitioner and team leader, Certified Scrum Master and Certified Scrum Product Owner, and had been an early adopter of Agile Methods since 2001, well before it became formalized in the Agile manifesto and so on. I'm glad to have Michael present this topic to us. Three Beacons, his company that he started is a leading provider of Agile training and consulting services. You can learn more about them at their website: www.ThreeBeacons.com. With that over to you, Mike.

Mike: Okay folks, welcome to today's session, today is already a good day. Usually when Hemant introduces me, he says something about me being a senior Agilest or something along those lines, and it's just his polite cultural way of saying, I'm a little bit old, and so I appreciate him not mentioning that again today.

Let's talk about the lifecycle of a user story, and I'm going to use this analogy of the lifecycle that we as humans experience. You can think of a lifecycle as a series of stages, in which something passes through during its lifetime. The thing can be a person, it can be a product, it can be a company, it can be a culture, and it can be a user story.

The word lifecycle implies these stages, and I've chosen these five stages, maybe there is some additional ones that you would like to add. But we are going to talk today about the conception of user stories, their birth, the growth aspects of a user story, the eventual maturation of a user story, and then yes the end-of-life.

Let's jump right into it, let's talk about the first stage: the conception, and how user stories are conceived.

Primarily user stories are discovered, or conceived through collaborative conversations. At this point from a conception perspective, we really only need the user story names, not yet the description and acceptance criteria, but I'll show you that here in just a minute.

Think of it as the names of the user stories, it's a good way to kick off your project, hold those discussions, where you are talking about the project vision and the ideation of it, and come up with the user story names because most of us or we all know that the user story is the decomposable work unit that we want to get to in our Agile projects.

Also I'll remind you that user stories are those that require stimulus from the user. Now the user can be human or not, it can be a user of your system or your application, or it can be another system, an upstream processor or downstream data marked or something like that. From when are they conceived, certainly during initial planning as the upfront vision discussion ensues, but also anytime you're discussing other stories, oftentimes discussing another story leads to the discovery of a new user story. Or even looking at a really big user story often called Epic, looking at that and breaking that down can be thought of as conceiving new perhaps more granular user stories.

Let's talk about some terminology here before we go much further. The classic definition of an Epic is, it sounds funny but it's a really big user story. According to Mike Cohn and his seminal book on User Stories Applied. Epics can also be thought of as a major business thrust or perhaps capability areas that the product needs to gather or maintain a market.

Now I'm going to follow the safe, the scale to Agile framework definition of how to breakdown Epics into stories and get there, so they have this concept of a feature. An Epic may be made up of multiple features, and features or services are groupings of functionality that can in and of themselves be broken down further into stories. Stories or user stories, these are specific usage scenarios that are granular enough to be completed within a single Sprint.

As I said before, in Agile projects the user story is the fundamental work unit, so you should always strive to decompose your system down to stories and not necessarily so much about modules or components or subsystems or anything like that. Talking about the user stories from beginning to end, you will see the different service aspects of your architectural layers.

Here is a real example of an Epic feature story breakdown that I worked on from a recent project involving a GPS navigation system. We'll call the Epic location

services. That's a pretty broad term, it's very high level, so anything dealing with location of the car driver would belong to this Epic. What are some features that location services might provide? A few examples, turn-by-turn routing, we are all familiar with that nowadays. Maybe some data centric features such as my favorite destinations and also maybe a trip computer, and then we break each of those features down into their stories.

You'll see for the trip computer feature are broken down into six user stories. In discussions about the trip computer feature, we identified these six user stories, and therefore conceived of these user stories through conversations about the type of functionality a trip computer needs, and in particular the goals of the driver using the trip computer feature. Again, focus on what user stimulus will be made and the goals of the user.

Now user stories can be conceived really at any time throughout the product lifecycle. If you've used this technique before, you'll understand what I mean by that, user stories seem to keep popping up all across the project schedule as we go. That is not a bad thing. As an Agilest we welcome change, and the art of discovering new user stories is definitely a good thing. Think about upfront planning, absolutely, user stories will be conceived. Discussing other stories, definitely, maybe in the middle of implementation some questions around how to implement it might see the need for an additional user story, during a customer conversation, etc.

Now let's talk about the birth phase. User stories are born when they are written, so whether you write the user story on a note card or within a tool, let's consider that the birth of the user story. Again I'm making a distinction between the utilization of the user story name, which in my mind is conception versus birthing the user story, which is the actual writing of the description and the acceptance criteria.

A few reminders here, be sure and keep the user stories high level. They are purposely high level to encourage conversation and the clarification of the requirement through conversation. Also think of it as capturing the gist or the essence of the requirement. Last point there, it's always a challenge to hurry up and get to Sprint one. At a minimum you need to have the highest priority user stories written out and birthed and the clarifying conversations around them in order to start Sprint one.

Here is the syntax of the user story you might be familiar with this already. Actually there is a raging debate in Agile land, half the Agilest believed that the syntax is useful, and the other half believe that its somewhat cumbersome, you

don't really need it. I'm in the camp that says it is useful, especially if you're new to this technique, use the syntax because as you can see looking at it, as a roll or who, that helps to set the context of the story.

Then explicitly identify what functionality you want as that particular person or role, so that business value. Identifying the business value is very helpful and helps make sure that all of your user stories will deliver that stated value. Then acceptance criteria, on the backside of the card or in the other section of your tool, thinking of acceptance criteria while discussing requirements is a form of test driven development. Effectively we are going to write the acceptance criteria or you can think of it as some high-level tests cases, we are going to write that first as we are discussing the requirement. It helps us …

Hemant: Mike, can I make one quick question?

Mike: Yep, go ahead.

Hemant: Showing these cards, which is syntax of the user story and the other side of the card is the acceptance criteria. Who is writing this, what role? Is this the QA writing acceptance test and product owner writing the story or the team effort or some combination thereof?

Mike: Right. The official answer from the Scrum guide is that the product owner, the Scrum product owner owns the responsibility of getting the user stories written. What that means is if a scrum product owner does not have helpful folks on their team, on their Scrum team, then yes, he or she must write all of them.

Now a little bit of history. When user stories were first invented in extreme programming by Kent Beck, he had the luxury of an onsite customer, and he gave the responsibility of writing the user stories to their customer. That worked out really great.

In practical reality the product owner typically does not write all of the user stories, but they should lead by example. If they lead by example, it's relatively easy to enlist the help of the Scrum team developers and QA, and even the business analyst and marketing and sales folks or internal stakeholders. You may not have access to direct access to your end customer, but if you do see if they would be interested in helping write the user stories also. Scrum product owner is the responsible person for it, but in reality it's a team effort and everybody pitches and writes.

Hemant: Who writes the acceptance test?

Mike: Whoever writes the story. It's not like one person writes the front of the card and then only QA writes the back of the card. That also is a team collaborative discussion. Whoever is writing the user story owns the responsibility of taking first stab at the acceptance criteria. Now you mentioned QA, it's always a good idea to get the QA person to review the acceptance criteria. Why? Because these folks are great.

They think differently than you and I do from a software development perspective. If you think your acceptance criteria is reasonably thorough, hand it off to a QA person, they'll find three or four really important things that you may have neglected. Again, it's whoever is writing the user story, but it's always a good idea to run the user story by the QA person.

Hemant: Sounds good.

Mike: All right. Thank you. Good question. Let's look at a real example here. This is what a user story would look like for one of our trip computer feature stories. I'm going to use the story start trip. As a driver I want to start my trip so that I can begin tracking my time and distance. Here is the acceptance criteria. I'll just read one of these, the middle one. When I view the trip computer screen, I expect to see the distance field changing as I drive. I also expect to see the drive time field changing in one second increments. You have the concept of the distance that I'm driving from a miles or kilometers perspective and my drive time from a one second increment perspective.

We've covered birth, sorry we've covered conception and birth, now let's talk about the growth phase. If you're a parent, I'm sure you will agree that your children just grow up way too fast than, user stories are a little bit different from that, some user stories follow a steady growth pattern, and they are discussed a little bit upfront and then about the middle of the project there is some growth indications and they mature later on in the project.

Some user stories though are asked to grow up very quickly. In particular the highest priority stories are somewhat forced into maturity really quickly. Some are fine never growing up. I don't know, maybe my wife would claim that I've never really grown up. These are likely the lowest priority stories or maybe the ones that are questionable or we are not really sure about but maybe those stories just never grow up.

To understand the growth dynamic a little better, we need to keep in mind the product backlog. The product backlog and Scrum is a seminal artifact, and it gives explicit guidance on which stories to develop right now, and which ones are

coming next and which ones are coming soon dot, dot, dot. Typically the product backlog consist of 80 to 90% of all the items in the product backlog will be the user story names. Throughout the project the user stories are shuffled up and down in the product backlog by the Scrum product owner. This contributes to the growth factor as the relative position dictates the level of conversations that are needed at any one point in time.

For example, stories at or near the top needs some really heavy clarifying discussions right now. The next set needs some good discussions pretty soon, the stories in the middle of the product backlog need probably a relatively small amount of discussion just for awareness, but not so much beyond that. Then the stories at the bottom don't really need much discussion at all until they start percolating up in priority. It's kind of a just in time type discussion model.

Hemant: Mike, can we take one more question? from the last page, if you can bring up that page?

Mike: Yep, go ahead.

Hemant: The picture in this page implies in order, priority order from top to bottom. To be able to put things at the bottom, you got to have somehow prioritize them. How do you go about prioritizing when they are still in the early stage of their lifecycle?

Mike: The Scrum says the product owner is responsible for managing the product backlog. That is true. The product owner has to make the tough decision and make a distinction between the 80th item in the backlog and the 81st item in the backlog. Often times the product owner has multiple inputs to negotiating the priorities. I would say the biggest one is business value. The product owner will look at the story names in the product backlog, and just based on their understanding of the customer needs and goals and the dynamics of the team, he or she will move stories that deliver the most business value up to the top or near the top.

Another thing to consider that the product owner might consider is, especially if they're familiar with the lean startup approach or a lean startup disciple is which user stories will give us the most learnings at the start of the project. Those would naturally gravitate up towards the top. Also the software developers may be lobbying the product owner to choose this story and that story and move it up to the top because if we as a team address that in Sprint one or Sprint two, then a lot of the architectural plumbing has to be put in place to make that happen because the end-to-end scenario goes very deep into the system.

The reason that's important to get done in one of the early Sprints is because if you take the time and effort to do it then, then a lot of the subsequent user stories just fall in place with that plumbing that you've already built out for the previous user story. There is a number of factors, I think Bob Galen in his book, Scrum Product Ownership, list at least 15 different factors, but I'm going to touch on what I consider the top three there.

Hemant: Can I ask a follow-up question on this prioritization, Mike?

Mike: Yeah.

Hemant: Are Epics also being prioritized at the bottom of the pile here. Some are bigger chunks and those may be Epics not yet broken down into features and stories, so are they being prioritized alongside the stories in the top or only stories get prioritized and epics end up staying as epics in some early stage?

Mike: You will definitely have the situation where you have the name of an Epic, and everybody agrees that everything within that Epic is lower priority than the other things. The product owner will place that Epic near at or near the bottom of their product backlog fully understanding that as that Epic starts percolating up, a breakdown of that epic into user stories must occur. That's also a growth factor that we are going to talk about here in just a minute.

However, it could be the situation where you have an epic of where most of the stories are low priority, but there is one or two that are medium or high priority. In that case you have to do that breakdown a little earlier than you might like to get to those higher priority stories and pull those out of that epic and prioritize those accordingly up near the top of the list. I've seen both situations, but generally Epics can be put at the bottom because they represent glums of user stories that just due to the area that they are in. Its lower priority than our pressing needs.

Hemant: Thanks.

Mike: All right. How can we help these user stories grow up? Again recall that user stories are purposefully high level, and as such they require clarifying conversations. One way to do this is you probably heard of this is the Three Amigos method. It's a collaborative discussion amongst three project rolls. Now these three project roles are developers, QA and either the product owner or the business analyst.

Now this method is not meant to be restrictive still, the more participants the better. Think of it as a minimal requirement that these three roles need to be played in order to have a good clarifying discussion. Again it doesn't prevent hallway conversations or lunchtime conversations about it, but definitely you want to strive to have these three roles. Also, again as a reminder, stakeholders, customers, marketing, business roles, definitely if they are available can be part of these discussions.

These conversations spur the growth of a user story to the point of becoming more and more clear, and discussing acceptance criteria, adding some acceptance criteria, modifying some others will help us identify when this story becomes acceptable to the customer. As these discussions ensue, the tacit knowledge is built up within the team. If you're a fan of Kent Beck's extreme programming Agile method, he has this concept, it's actually a principle called Whole Team, which emphasizes that each team member is valued and belongs to something larger, namely the team. These Three Amigo conversations can help emphasize this concept in the spirit of inclusiveness.

Now let's go back to the start trip user story, and after a Three Amigos conversation we might have found a few clarifications shown in red. The business value for start trip has been modified to include a clause, which further clarifies it. And it says so that I can be in tracking my time and distance, in order to accurately fill out my expense report. Then as we held these Three Amigo conversations we found an additional acceptance criteria. It reads, when I view the trip computer screen, I expect to see buttons for stop, save and reset.

Guess what? If we do not yet have a user story for stop and save and reset, then we better add those user stories right now. As you can see the user story grows through these conversations and we add notes to the user story, and it's basically firming it up.

Another important growth factor is when the user story receives an estimate, you can use story points, probably the most popular or T-shirt sizes. Even hours or man days, doesn't really matter, but estimates become really important in Sprint planning. They help the team make a righteous choice for the upcoming Sprint. They're also very useful in release planning. They help the team predict what will be there in a future release. I'm always amazed in my Agile projects when we do a release plan how reasonably accurate that is.

I always hark it back to my waterfall traditional experience and remind myself and the team that waterfall really doesn't have anything similar to measuring

velocity and using that to reasonably accurately predict an upcoming release, so big advantage to Agile right there.

At times a user story requires some experience that the team just doesn't have. In this case the story grows from the additional learnings of what's called a research spot. This may involve study of some reference code, it may involve some Internet research, it may involve development of a prototype for downloading a new technology stack and playing with it to become more familiar with it. Research spikes are intended to be time constrained and exploratory. Typically a single Sprint with a subset of your team members or I guess in the rare case your entire team.

I like to equate this to a growth spurt, it's analogous to that. In that we know a lot more about the story than we did before. Now the story is prioritized. We have good clarity of requirements based on the Three Amigo discussions, and/or research spikes. The story has an estimate. Now we are ready for implementation. When the story is placed into a Sprint backlog and broken down into its respective task, the story experiences a significant growth event. That's because now all of a sudden we understand a lot more about that story, and in particular the how aspect, how the implementation is going to proceed or these are the tasks that are needed to implement the story.

Hemant: Mike, can we take a couple of questions from the stage?

Mike: Sure.

Hemant: One is on story estimate that you just talked about. Does that estimate when it comes back either in story points or T-shirt sizing or man hours as you said, any of those units, does that play back into the prioritization in the sense that big stories.

Mike: Yes.

Hemant: Get moved down, small stories move up. Let's take that question and I'll come to second.

Mike: Yes, yes. It absolutely does. The size, a relative size of the story does. It is a contributing factor to prioritization. Primarily it's about, okay, is this story right-sized, so that if the team chooses it, it could be done within a single Sprint? If it's not right-sized, then it's deserving of some more conversation to break it down into its sub user stories, if you will, or it's the classic Epic to user story breakdown.

That's pretty rare that you might find a user story that is end-to-end in nature, that does not fit within a Sprint. If it does, there is some pretty classic techniques of how to deal with that, basically identify demark point and implement the top half of the story in this Sprint along with the demark point that returns a success code. Then in the very next Sprint implement the bottom half, and throw out the demark point and do the integration, and then demo the full end-to-end slice things like that. But for the most part estimates do contribute to the priority.

Hemant: The second related question that you started to touch on anyways is when the estimation happens, how do the team know when it is too big and one definition you already gave on too big is, if it doesn't fit in the Sprint. But aside from even if it fits in the Sprint, is there any guideline on what story estimate should be considered as too big and therefore worthy of making down?

Mike: It's really an art. There is no hard and fast rules other than it should fit in a single Sprint. The other advice I would give you though is as much as possible, you want your user stories to be a thin functional slice of end-to-end logic throughout your system. For example, if the user presses the play button, you should have a user story called Play Video. Ideally that then functionality will be able to be modeled and implemented within a single Sprint. The only rule that I'm aware of it must be, it should be able to fit into the Sprint, but then you can think of the user actions and try to keep it slim and thin, so that a user story can be done in three to four days.

There is some techniques for how to get there. Often times when you're dealing with alternate paths and variants and derivatives, you think, "Oh my gosh, even though it start video player, there is no way that can be done in a single week or single two week Sprint." But what you can do is pull out some of those alternate paths and make them their own user stories. Maybe it's something like handle the exception cases for when the video player does not start or something like that. There is ways you can bust out the full set of functionality into separate stories just to ensure that you are at that right granular level. Again, a lot of it is art and the more you experience this, the better at it you'll get.

Hemant: Mike, let's take one of the clarification you talked about a few minutes ago about using the top half or the bottom half to break a story down. Does that go against the Sashimi Slicing kind of a thought process around or you are implying something different?

Mike: Yes. It absolutely does go against the Sashimi. It's the rare case, it should never be the norm. I must admit oftentimes I'll engage with the team new to Agile, and they think that what it is, is just defining those demark points and writing code to

that then demonstrating that. But that was never the intent. It was intended to be a fully functional end-to-end slice. It definitely stay within the spirit of Scrum, and it's effectively a fundamental tenet of Agile also. I was describing the extremely rare case where even a thin functional slice just due to complexity or size reasons cannot be done in a single Sprint. But don't let your team opt out for that, keep them in the opt-in mode.

Hemant: That's good clarification. Thank you, Mike.

Mike: Good. That was an excellent question. Let's keep going here. We talked about the breakdown of the task, and that that's a significant growth factor. Let's go back to the start trip story. Here are some potential task for that story. As you can see these also contribute to the growth of the story from an implementation perspective. Look, there is even set trip timer, design a trip database record, calculate the distance, dot, dot, dot.

We've gone through conception, birth and growth, now let's talk about maturity. In my mind user stories mature when they were implemented. We focus on working the task within the Sprint, and also those are identified in the Sprint planning meeting. Often times we'll find new task as we are implementing, so we are going to mature that user story by firming up the tasks and actually implementing them. Anytime we implement, we have to crosscheck that against the definition of done or we should. Many agile teams use the definition of done to know really when a user story is done. Becoming done is a significant maturation step.

Examples of definition of done for a user story might be the following things. The code adheres to our coding guidelines. The code has been reviewed by at least one other person. The code builds automatically without air. We have automated unit tests that run green for that user story, etc. Many times the user story must, and the implementation of it must morph to achieve the definition of done.

Now let's go back to the start trip story, and then let's see how that matures. Focus in on the red rectangle there. We had written previously, "I also expect to see the Drive Time field changing in one second increments." During implementation we were showing this to the QA person, and they were about to give it a big thumbs up, and then she asked a very important question. Does Drive Time start immediately when I select the start trip button or does it wait until I actually start driving?

We had a big oh moment or oops moment, so we checked with the business analyst, who confirmed that Drive Time should start only when driving. This caused us to do some rework inside our code and we went back to the user story and busted out a previous acceptance criteria into a separate one of its own, that says when I began driving I expect to see the Drive Time field changing in one second increments. Like I said it involved a code change and then finally after that the QA person was good with it, and we got their thumbs up within the Sprint.

Let's talk about demonstration. Demonstration is a culmination of a lot of hard work, effort and conversations. When demoed the user story experiences a significant maturation event because there is going to be, there is likely to be a feedback on that demonstration. There can be a thumbs-up or thumbs down from the QA person or the product owner. That's also a maturation tag of sorts. If it's thumbs down, then there is some rework required and some additional maturation towards doneness, and except being becoming acceptable to the product owner and the QA folks. For the start trip, we as a team we would demonstrate that the Drive Time field does indeed start incrementing when the car begins moving, but not before.

Deployment to the field is a final maturation step, and which is actually getting the user story into the customers' hands. At this point you may receive feedback from the customers or even a few bugs, and guess what, fixing bugs or improving the user story is an additional maturation factor. Just to summarize here, we've got task breakdown, we've got implementation, integration and testing, we've got definition of done assessment, and we've got receiving feedback from demos and deployment. All of these contribute to maturation.

Finally, let's discuss end-of-life. There are various reasons for a user story to experience end-of-life. A couple of those are the user stories not really needed anymore, maybe the product is decommissioned, maybe the project is canceled, etc. Whatever the reason is, there is no need to mourn. In general it's just a natural progression of systems development. This certainly applies to user stories also.

Let's conclude here, I believe that …

Hemant: Let me clarify the terminology in the end-of-life that you are saying. When a story goes through the entire cycle and they are deployed in the field and users are using it. So long as users are using it, it is a still a live story.

Mike: Yes.

Hemant: In production views, right.

Mike: Yes.

Hemant: The end-of-life only happens if it has been removed, functionality is removed from production.

Mike: Yes. That is exactly right then, and thank you for introducing that because this slide probably needs a statement to that effect. The user story, if the system is still out in the field and that feature or user story is still being utilized by customers or clients, then absolutely it's, it's still alive out there and we can be proud of that. At some point may be the product hits its end-of-life cycle or its decommissioned or, and oftentimes it supplanted with a new evolution of the product. At that point you might consider at the end-of-life for that user story. Excellent clarification, thank you for that.

Hemant: Sure

Mike: Let's conclude and wrap-up. I believe user stories follow a lifecycle analogous to human life, and you can think of it through those five phases. I think thinking of user stories throughout these lifecycle phases can help us all achieve a more in-depth understanding of the user story technique.

With that, here is just a brief commercial about Three Beacons. We specialize in Scrum team training, Agile Immersion, user stories, as you might expect. We have a lot of experience in that, and even extreme programming, if you're looking for how to develop high quality software. We also do consulting. That's the end of it. I'm going to turn it over to Hemant here to step us through a few Synerzip slides. Hemant?

Hemant: Sounds good. Thanks Mike. For audience, we'll take questions now, I already have a few questions in the inbox, but while you guys think through the content and keep sending more questions. In that meantime I'll quickly introduce Synerzip and then we'll come back to take some really good questions. Mike, if you can advance the pages, so I can talk through Synerzip introduction for audience who have not been exposed to Synerzip so far.

We are in a nutshell we are a software product development partner company for small to midsize software technology companies, and we help all our clients essentially accelerate their product roadmap by putting together a team of high-caliber software professionals, which is dedicated team for each client and works as an extension of in-house team, and we are following Agile practices and just

good sound software development practices. We help a client reduce the risk that is inherent in building new software. We also reduce the cost by leveraging our offshore India-based software development center.

Typically we are able to provide a 50% cost advantage to a client, while they are getting these benefits of acceleration of product roadmap and risk reduction and cost reduction, they still maintain the flexibility, long-term flexibility, our clients do by having the ability to turn that operation or team that is working for them at Synerzip into their own captive team whenever they feel the need to do that. That's what Synerzip does all our clients or long-term clients, and this next page shows a glimpse of some of our clients. A few large companies but mostly midsize software companies, that's the focus. Typically venture backed growth stage software companies.

With that let me give a little plug to the next month's webinar in our ongoing series of Agile webinars. Our next webinar next month is going to be on Analysis on Agile. That's a very difficult topic for Agileists to focus, how much analysis is too much and how much is adequate for the purposes of good software development, so we'll cover that topic next month on April 15th. Do plan to join us presented by Kent J. McDonald, is definitely an expert on the topic.

With that let's turn our attention back to this topic of lifecycle of a user story and the questions that go with it. If you are ready Mike, I'll take some questions that are in the queue.

Mike: I'm ready. Sure.

Hemant: The first one is, is the role of QA going away because of Agile? The point is often Agile practice seems to talk about test-drive development where the whole of development is involved in testing, so the specific role of QA does that seem to go away in the Agile context.

Mike: The answer is no, emphatic no. As a matter of fact, the role of QA in an Agile organization is much elevated. Think of it as this. The role of a QA person in an Agile team, it changes from a traditional role. In that the QA person is the developer of test case automation. They are focusing in on the acceptance criteria of the user stories and figuring out a way how to all automate the testing of the user story commitment for that Sprint.

They are working hand-in-hand with the developers, and they are watching their progress and doing the best they can to anticipate what is going to be developed within that Sprint and the anticipation they will turn into test case automation.

You'll hear their phrase, test-driven development, and mostly that's about unit testing from the developer perspective. Make no mistake, quality starts with the developers, the software developers writing the code. They should be writing unit tests either before or hand-in-hand with their production code.

Now the QA role in an Agile team is to automate the behavioral testing, which is typically described by the acceptance criteria in the user story. In order to do that, they have to work very closely with the developers. It's no longer this traditional world where the QA team is a separate team and the product is pitched over this wall of uncertainty is what I call it. As software developers, we tell the QA folks good luck, you only have three weeks to test it before launch. That's when disaster hits.

Just in summary the role changes, the responsibility becomes more of a test case automation person, and in effect it's a specialized developer, is a way to think about it.

Hemant: Excellent, thanks. Let's take next question which is, when you have an existing system in place, so by your lifecycle definition it has matured user stories, these are features being used.

Mike: Right.

Hemant: You are now modifying an existing user story, so I guess there is some enhancement request there. What does the best practice create a new user story and retire the old, or modify the existing story and represent the new requirement with a modification?

Mike: A really good question. I would think a best practice is, if you're really, if that code base for the existing system was developed using the user story approach, I think the best thing to do is go back to the original user story and modify that. Again, it's a growth factor, modify that, the description and accept as criteria associated with that, and include that as a record in your current project or your support maintenance approach. Then obviously do the coding changes.

Now oftentimes existing systems were developed without using the user story approach, then my advice would be write a brand-new user story, but keep it focused on the change. Don't feel like you have, you are obligated to go back and describe everything about that user story. Try your best to just keep it focused on the change.

Hemant: Excellent, thanks, a somewhat related question but may warrant a separate answer. What's the value of keeping a user story after its successfully being deployed to production?

Mike: The value is that, this is the user story in action. This is the end goal of all of the conversations and design thinking and implementation and unit testing and behavioral testing and stress testing and to get it out to the field. That's the culminating highlight of all of that work. Most software systems once they get out to the field, they have to be supported. It's always good if your support team has the mental model of this system is a collection of user stories that are currently active out in the field. As bug fixes are needed or improvements are needed, let's model those and drive those based on the user story.

Hemant: Thanks. Let's take another question, Mike, which is on this done criteria. Can you talk a little bit about the variety, what are reasonable, I know you had a picture or a draft or a table in your slides, but a good done criteria, should it be deployed to production or should it be code complete, some guidance on what's inappropriate in criteria to be using?

Mike: This is always a discussion amongst the team, you definitely want the team to come up with this, and it's not mandated from above because that way the team will fill ownership of it. Now over the course of time, you may have from project to project a consistent definition of done coming up, but always have the team review it and see if they can recommit it to it. But at a minimum, at a minimum it needs to be along the lines of, the code adheres to the coding guidelines. You should have coding standards, document and coding standards for JavaScript or Java or Objective-C, whatever programming language you are using.

There are always some constructs to avoid, some techniques to do instead of one way versus this way, some stylistic options because you do want consistency of the code. At a minimum the code should adhere to the coding standards plus the code has been reviewed. If you use a pair of programming, that's reviewing as you go. If you don't use a pair of programming, then establish a buddy system where you and a buddy review each other's code and give feedback to it. Also the code for the story to be done, the code must be source controlled in SVN or whatever your repository is.

There is a bare minimum there, which are good practices that every software development team should use. You can start building on that. If you don't yet do unit testing, but you would like, then add the automated unit test run from a continuous integration perspective as part of the definition of done. That the entire system goes green with your new user stories integrated, things like that.

Then step it up over time. Introduce the concept of complexity analysis. You know what, I've read methods in classes that go on for 30 pages long. Shame on all of us for doing that. The cyclomatic complexity of that method would probably be 200, which means I as a normal human being, software engineer trained cannot understand that method.

Keep your method short, and they should all fall under cyclomatic complexity of 10 or 15. Introduce that topic is a way to step up. Here is another one, memory leaks. My gosh, if you're not checking for memory leaks, that's probably the number one reason applications crash, right. Just successively add to your definition of done, introduce the tools that will let you do that relatively easy, so that people don't feel this huge pain of having to do all of this over at work, start with something simple, but then add something every project after that.

Hemant: Thanks Mike. Let's take another one which is if at the time of iteration demo, the story is not accepted by the product owner, what happens in the lifecycle, does it get now logged as a defect or what happens?

Mike: It's like asking the beautiful girl for a date and she rejects you. You have to, that story is not, it's definitely not demoed, right, because the product owner has not accepted it. It's a try again next Sprint, so it carries over to the next Sprint. Again, the reimplementation of it, or the additional items that seem to be missing, all of those in the next Sprint, all of that activity contributes to the growth factor. Actually it's a very good point because you can think of, it has to be a certain age or a certain growth level to be able to be demonstrated in the Sprint review. If it's not up to par or not at that growth level, it gets pitched over to the next Sprint. Also as a reminder, you want that to be the rare case, not the norm.

Hemant: But it maintains in the lifecycle still as a status of a story.

Mike: Yes.

Hemant: That is continuing to mature.

Mike: Yes. Yes, exactly, it's still. Yep.

Hemant: Let's take one final, maybe a final question. Are there any other techniques you suggest so you, beyond what you've covered here in this webinar for prioritization, for example, the parts, some followers of Agile talk about this notion of value points, which is ascribing certain set of full value points to a software project or a product, and then using that relative distribution to do prioritization as a way.

Mike: I think actually that is a good technique. I personally have not ever used it, but I have read about it and it looks very interesting. I like the idea because every user story in your product backlog or every workout, your product backlog has some associated value. If you can put a numeric rating on that, then that should help the product owner prioritize those. If something has a 30 value rating versus a five value rating, then absolutely.

The question becomes how do you come up with that numeric value? Is it primarily driven by business value or is it an amalgamation of everything that we've been talking about, including business value, how much learning will we get from that? Is that architecturally significant? Is it from a time-based perspective do we want to do this one first, dot, dot, dot? Maybe it's an amalgamation of all of that thinking, which can only be of a big benefit.

Yes, definitely, I would advise the value point system, but I would also advise your listeners to read more about lean startup and how to treat your project as a collection of assumptions and reword those assumptions into hypotheses. And then what are the product backlog work items that can answer that hypotheses either in the positive or negative sense, because in the lean startup community, it's all about learning. So which of these user stories will help us learn the most?

Hemant: That sounds like an excellent topic for one of the upcoming webinars, Mike, doesn't it?

Mike: We'll have to talk.

Hemant: That is definitely a good topic for one of our webinars that are queued for next few months, but thank you so much Mike for covering this topic of Lifecycle of a User Story.

Mike: Sure. You are welcome.

Hemant: Excellent, Mike for covering this topic and thank you all for attending on the audience side. Stay tuned for next month, the April 15th we'll have our next webinar, Analysis on Agile. Thanks. Have a great day. Bye.