Transcript: Agile Adoption Lessons
Free Webinar on September 22, 2014

Hemant Elhence: The reason we picked this topic this time is, as you all know, Agile software development and Agile in general has been well over ten years old. Manifesto was signed in 2001. So technically 13 years old officially of Agile practice, but still even though there's a widespread adaption of Agile and it's fairly mature in many companies, it's defined that many companies have challenges around adaption of Agile, and we thought this was the case study and the experience that we'll have Jim Smith share from his company. PDX example is a great one to share as the series of lessons learned from adaption of Agile which many of them you're going to find resonates of many of you.

That's the topic today. In this situation that we start up, usual complexities of distributor development and so on that we'll cover. It will be all excellent lessons and I'm glad to have that covered today. By way of logistic, we'll have about 45 minutes or so of presentation, and we'll take some questions along the way and they make sense but we'll still reserve another ten to 15 minutes towards the end for taking general questions.

You guys keep thinking about your questions about challenges you see and what you want to ask, and keep sending them into the Q&A panel here. I'll monitor the questions and I'll inject some questions along the way, and then others we'll take at the end. We may not be able to cover all the questions but we'll try our best. With that, let me introduce the speaker and also we will make recorded version of this webinar, as well as the PowerPoint to the document available to our website in a couple of days after the webinar.

With that, let me introduce our speaker Jim, Smith. Jim is the Senior Vice President of Innovation and Trends at PDX. PDX is a retail pharmacy software company, 25-plus-year-old company, and very successful probably number one player in their market category. Jim for 11 years was overseeing software development teams at PDX and various sizes and shapes of these things before he decided to jump on to Agile way of doing product development for all of his four product lines which was around 250 development people at PDX.

Jim was introduced to PDX to introduce PDX to scrum, a form of variation of Agile just about ten years ago now. He will share his perspective and I'll let Jim provide more color on PDX and his specific role at PDX. With that, over to you Jim.

Jim Smith: Thanks, Hemant. I'll get started now. First, I want to tell you a little bit about myself just a tiny bit. First of all, I'm not a consultant. Hemant just mentioned that I've been at PDX for a while. I've been there for 22 years. I hope to retire there someday. Not that there's anything wrong with being a consultant or anything like that, but often times I detect when I give this talk to various companies that they think that I'm going to come in and try and shake things up, and try and change the way they do things and that's not the case.

Basically what we just like to share information that we think is going to be useful to the community. That's really the whole purpose of this thing. I'm a developer by trade. I'm a Java developer. I still like to write some Java code but I've been in management for the past 13 years. This says 11 years. We've created this presentation about two years ago. We didn't really have the need to update it much. It already has quite a few lessons on it, and it will be a little bit of a challenge to get them all in within 45 minutes. We're going to run through this thing. The format for this presentation is pretty straightforward. I'm just going to go through a list of various miscellaneous lessons that we've learned at PDX in our implementation of Agile.

One thing that's interesting is that we're still learning. Even though I handed the rest of software development over to one of my peers two years ago, Todd Crosslin with PDX, we're still learning. We're still making changes to the process. It's something that never ends, but at the time two years ago I oversaw the development of four pretty complex product lines, retail pharmacies, software product lines and that included about 250 programmers, QA testers, software architects and database administrators. What we call our design team which is that's basically the business analyst that also designed the software. They do more than just plain business analyst do but that team reported to another writers within PDX.

Jim Smith: So a little bit of context before we go to the individual lessons too, I want to tell you a little bit about PDX Incorporated. We are, as Hemant said, the leaders in the retail pharmacy software industry. Some of our customers include Target, Safeway, Albertson's, Kaiser Permanente. We've got a bunch of big retail pharmacy chains as customers.

In addition to a pretty large number of independent pharmacies, something that a lot of people don't realize is retail pharmacy is extremely complex. In addition to managing workflow within a

pharmacy, our system also does pricing on prescriptions and the pricing schemes are very complex. It manages drug inventory. It does clinical checking on prescriptions which itself is very complex piece. It manages online adjudication with insurance companies, and then the processing, and the responses, and then it interfaces with lots of other different systems including more sales systems, ITR systems, pharmacy systems and so on lots of different systems, and the landscape is constantly changing. Our customers are always probing looking for new ways to get revenue with the pharmacy as their margin shrink in time.

And so what we had going on about ten years ago was … Actually a little longer than ten years ago, we had a legacy system that had been around for about 20 something years. It was an old text-to-date system. In fact, it's still in production. It's in place in thousands of pharmacies but we knew even 12 to 15 years ago that we needed to replace it. We needed to update it with new technology. We embarked on a new project we call the Enterprise Pharmacy System or EPS.

So just to give you a quick idea about where we were, I had the experience to the point that I took over that team as a software development manager. I had about five developers working on my team and I managed them really well. I thought I ran a very tight shift. I knew what every single one of those guys were working on it at any given moment. All communication funneled through me, the manager of that team. They didn't even really have to talk to anybody outside the team, not much and no one outside the team really needed to speak to them. All communication came through me. We delivered the software, what was ready, and it all seemed to work well enough.

However once we started on the enterprise pharmacy system, building a brand new pharmacy system from scratch, the old techniques stopped working. For several months, we spun our tires as we try to get this pharmacy system off the ground. All of our traditional techniques weren't working when we had dozens of programmers and QA testers working on the project. We knew we had to resort to something different.

A developer came to me one day, and he knew that we were very stressed out because we hadn't made a lot of progress yet. We did certainly didn't have anything we could show to anybody to inspire confidence outside of our group. And he said, "Hey, I heard about this thing of scrum. Maybe have a look into it. I think it worked really well with our company." And I said, "Yeah, yeah, yeah whatever. I don't have time for this really," but then, he really tracks the issue, and I started to

listen, and eventually I said, "You know what? That might work but I don't think we're going to call it scrum. That sounds too gimmicky. We'll just start developing software that's on the way."

We spoke to a lot of other folks on the team, a lot of the other stakeholders and they started to buy in, some of them were reluctantly than others, but then once we got things going, it was to me like seeing the pyramid is for a lot of people or seeing the Grand Canyon for the first time, it blows your mind completely. There were a lot of things that started to happen once we made the switch to Agile development that it never happened before and it really was truly, for me, amazing. Some of the things we saw, I'll talk about them in these lessons learned.

One of them that really amazed me was how the amount of motivation I started seeing coming out of the developers. When I say, "developers," I just mean programmers. I mean quality insurance testers, UA testers and design folks too and others, but it really blew my mind how those folks, their motivation levels just went through the roof and I had to in the past, I no longer had to ask people to work overtime. They just worked overtime if need to, and if they didn't need to work overtime, they didn't. Getting the opportunity to select what they were going to work on and then to commit what they're going to work on. Having good folks, we got great folks, they want to meet their commitments, and so they do everything they can to meet them.

There are a lot of really amazing things going on when we adapted Agile development and we started then with scrum. At the same time we experienced great success. By the way it wasn't just within the team, it's also outside the team. We started building confidence not only within our internal business stakeholders but also with our customers. They started seeing real progress. We were able to demonstrate the software at the end of every iteration and it really built their confidence quite a bit. Even though it was wonderful, we did learn a lot of lessons along the way and we're continuing, as I said, to learn lessons. I'm going to jump through a whole bunch of those lessons in no particular order. Let's jump right into that.

Lesson number one, everybody thinks that they're already Agile. In fact, I can remember at the time we decided we were going to do scrum, one of the reasons we decided was okay, it was because we felt we are doing a lot of the things that scrum dictated we really ought to do, that the approach dictated we ought to do. For example we really thought that we were doing great public collaborating. We never wanted to imagine

that we had developers in separate silos working on separate things, but we really were.

That's something you tend to run into when you're adapting Agile. You're going to run into this folks that say, "We already do Agile," but there's some important distinctions that if they're not doing them, you're not really doing Agile. For example, if development managers are just still handing out tasks to developers, that's not Agile. If you've got people basically saying, "Hey, we have iterations in development in the last two months, and when we get to the end of a sprint and iteration of development, if the work is not done; well then, it's not done, and we just take a deep breath and try and work into the next iteration." That's not really very Agile either, but that's one thing that we learn. One of the forms of resistance you run into in adapting Agile is individuals believe that they're already Agile.

Let's move on to lesson two. Enabling developers to commit has spectacularly good effects. As I was saying earlier, when you tell a developer, you're going to select which you're going to work on. I'm not just talking about programmers. I'm talking about QA testers and so on. They own it and they manage their own overtime. They drive their own teammates. They escalate.

They own it. That means that they take more of an interest in writing the code. They don't just code to the specifications. They start getting involved. Those walls between silos start coming down because they really feel as though they own it. When you feel that you own it, you really wanted to be something good. That one of the seeing-the-pyramids moments for me is seeing one and it seems like a no-brainer but a lot of development organizations don't do it. I have to confess, we weren't really doing that before we started adapting Agile fully. Giving them the ownership has great effects.

Another one that's functional already is, they manage their own overtime. Gone are the days that you have to say, "Hey, guess what guys? We're going to have to work late tonight to get to release stuff." You're going to see the team doing it themselves.

They drive teammates. They drive each other. There's nothing worst when you're trying to meet a deadline when you got a member of your team that's not also trying to meet the deadline. You'd be amazed at how developers will start to drive their own teammates.

As you can see we're moving towards when you adapt Agile the team running itself and the manager having to do less driving. We'll talk more in a minute about how the manager gets other opportunities to do great other things when the teams are running itself.

Another something that the team does is they escalate. When they're trying to meet the deadline, when they're trying to complete the iteration, complete all the work that they committed by the end of the iteration, then you're going to see them starting to escalate right away. They're not going to sit around and say, "I hope that problem takes care of itself." They're going to bring to your attention more rapidly the things that are standing in the way of them getting the job done. These are all some of the great effects that giving developers the opportunity to commit has on your team.

Lesson three, managers get to do good things. I heard development managers as they're embracing or as they're adapting Agile, I've heard some of them express and I'm not just talking to one or two, I'm talking about many of them including me, express the concern that "Hey, once the team starts running itself, what am I going to do?" The answer is there's still plenty to do. In fact, it frees you up as a manager to do so many other great things that really ought to be happening on a team that don't happen on teams often enough.

One of them is coaching and mentoring. You get the opportunity to take all your years of experience as a software developer or as a manager and to use it to code some of the folks that are coming up in your organization that are working their way up the ladder. You get the opportunity to mentor them, to coach them.

It gives you the opportunity to make strategic organizational infrastructure enhancements and fixes. Let's say for example, you want to switch from one bug tracking system to another or maybe in your previous life before Agile you didn't have time to do it. There's just too much to do. You're working on getting that release out the door. When the team is running itself, you get the time to do all those things that really should be happening for your team. Your team's infrastructure technical data, you're knocking that out. You're getting the opportunity to address it.

You get the chance to audit. You get the chance to dig down and look at what your team is up to making sure that they're doing the right thing. They're abiding by best practices. You're able to get and stay plugged into the business. I can't tell you how many development managers have

seen they're completely deep coupled from the business folks that they're serving. Often times the reason is, I simply don't have time to engage with them. I don't have time to talk to them. When you're doing Agile not only do you have the time to engage with them but it's something that you really have an interest in doing in order to make the product as good as possible.

It also gives you the opportunity to keep your flip soldiers educated. Again, something that I've seen in a lot of development organizations is that development manager is leading a form of complacence. They are basically not driving their team to stay current on technology. When the team's running itself, you can take it upon yourself as a development manager to ensure that developers, your technologists in general are getting regular doses of update information on what's going out there in the industry.

Java8 is now in GA, for example. If you have a team of Java developers, are they up on what Java8 is all about on the features that it has and so on. You get the opportunity to do this sort of thing when you're doing Agile.

You're able to give them morale, the maintenance, and care, and feeding, that it deserves. A lot of development managers have seen them do this thing. They're not able to address morale problems because they are too busy trying to get software knocked out the door.

Then finally, you are able to talk to other people. You are able to actually engage with some of your peers to find out what's going on with other teams so that you don't duplicate effort or so that you don't reinvent the wheel. Also, it gives you the opportunity to coordinate. Talking to each other is one of the benefits of Agile. Managers get to do that with each other; whereas previously, in a lot of cases, they simply don't have time is the claim that they've got.

Hemant Elhence: Jim, can we take one question while you're on this lesson? Did you guys do any education or formal training on Agile practices or Agile overview for managers and the executives before you started?

Jim Smith: That's a good question. Initially, we did not. We basically read some articles on how to do scrum and we said, "Okay. Let's just have a meeting and we'll explain this to everybody how this is going to work." We did that. We answered a whole bunch of follow up questions. We made adjustments along the way with that first print.

One thing I really wish that we had done that we did about a year-and-a-half or two years after we started doing this is that we got some help from outside. We had someone come in and give managers and individual foot soldiers training on scrum what it's supposed to be all about. I remember that guy came in and wanted to know a little bit about how we do things at PDX. I said, "I'd rather not tell you that. I don't want to poison the well. I want you to just come in and tell us about how you do scrum, and not necessarily how you do scrum but just tell us what scrum is, what it's all about." He did that and that helped it fill a lot cracks and it helped us to correct a lot of bad things that we were doing.

Hemant Elhence: As a recommendation to our audience here, would you suggest that if they're struggling or trying to adapt Agile to do up front education or some overview for executives and managers?

Jim Smith: I would recommend that because it's highly available out there. When we were doing this, it was a little known thing. You can mention the words scrum or Agile to a lot of the development managers and they didn't know what you were talking about or it was just a nebulous idea to them, but these days, those guys are everywhere. Agile coaches are very available. You could hire someone to come in.

However, I do want to say that in some organizations I've seen, there's just a natural suspicion associated with bringing somebody in from the outside. The usual issues like they don't understand our business, they don't know how we do things, we're special and so on. If you're in that kind of situation, I recommend that you just read some articles because again articles are plenty out there, and you try and do it on your own. Then if you struggle for a little bit, bring in someone from the outside. I do recommend you bringing someone from the outside regardless but some organizations might not be able to do that because of the constraint I just mentioned.

Hemant Elhence: Thank you.

Jim Smith: Sure. Lesson number four, write good user stories. Good user stories are a great thing. If you don't know, user stories basically just a single sentence that encapsulates a requirement in Agile. An example might be. "As a pharmacist I need to be able to document the reasons I'm overwriting the clinical check in order to create a complete clinical audit trail."

Good user stories are a great thing. They tell a story in just one sentence. When you get good at writing them, they don't take very long to create. We ran into some resistance within our organization when we first decided to start writing user stories, the claim was, once again, we're not going to have time to write those things, but we pushed, and eventually our team started writing them. Now, we do it all the time. It's just given that everything basically revolves around user stories.

You have to be careful. Sometimes user stories encapsulate not just requirement but design. Sometimes you'll see someone writing a user story that says something along the lines of, "As a pharmacist I need a button that does blah, blah, blah." You don't really need a button. What you really need is to fulfill a given business requirement. It doesn't matter whether it's a button, or a radio button, or a text field, or a check box or whatever, it's going to depend on the design of the application once you get there, but there's no need to incorporate that. In fact you should not incorporate that into the user story.

Another great thing about great user stories is that they make everybody understand the big picture. They're not great tool for breaking those walls of silo walls between teams. Individual programmers, QA testers that are not on the front lines with the business, they start to understand why you really need this piece of functionality. They start to understanding the big picture and that just produces great things, great software.

One little test that I think is useful for testing a user story to see if it's a good one is what I call the no system litmus test. You just ask yourself the question what if there were no software? For example in a pharmacy, there was a time when there was no software or no computers in pharmacies. They used typewriters and they wrote things out of paper. At that time, the user story that I just mentioned it still had applicability. Even in the pharmacy in 1950, there was still a need for a pharmacist to overwrite any clinical checks that popped up in his own brain in order to create complete clinical audit trail.

If you find yourself needing really to understand the whole system, to understand the user story, the next usually is a bad sign. Anyway, you can easily go on for a day or two talking about user stories. That's just one little lesson. Good user stories are a great thing.

Lesson five, the industry feels that preplanning is necessary. If you go back to ten years ago and talk about Agile development, part of the whole Agile development was basically death to waterfall development.

If you find yourself doing anything upfront, that's usually a bad sign, that's not Agile. That was the belief ten years ago in the industry.

If you show up for a kickoff meeting and you got everything already figured out, then you're not getting the job done properly because you're not giving everybody the opportunity to weigh in on the design or not just to weigh in but to actually help with the design of the software. Just in practice, it seems that the industry now believes that preplanning is a fundamental part of the process. I have to say even at PDX we do preplanning these days and that basically just means before a kickoff meeting happens we've already figured out a lot of things. It makes the kickoff meeting go more smoothly.

Now, as far as my personal editorial goes on that, I would say that I probably agree with a lot of the folks that say that it's not really necessary but that's assuming that everyone on your team is 100% Agile-minded and that they're capable of diving in altogether to help design the software. In a lot of cases this is not possible just because the software may have such a specialized use that you do have to have someone with more knowledge and experience about the product, figuring some things out before the kickoff meeting happens. This is the consensus of the industry even though I personally don't completely agree with it.

Lesson six, TDD is a way of life. For those of you that don't recognize that abbreviation TDD, it stands for Test Driven Development. Basically, I'm not going to try and explain all of the test driven development on this webinar. I recommend that you Google it if you don't know what it means but it is really a very important part of Agile development.

As I mentioned, way back in the day when I was managing a team of five developers, the software was ready for delivery. We delivered it when it was ready, once we had tested the heck out of it and verified that all the bugs were gone, but with Agile, you can't do that so easily. You got to declare that you're going to be done on a given day. That the sprint is going to end whether you like it or not on this particular day.

To ensure that you got good quality, you got to do test-driven development. You got to make sure you write your unit test up front. Again, I'm not going to get in to what TDD is all about but I do have to say that we learned that it has to be an integral part of how you do development. You got to adapt it wholeheartedly or else you're not going to see good quality or you're going to have to extend, you're going to have to say, "Well, our sprint is done but we're going to need another

three or four months for this thing to bake," as we repeatedly fix bugs and regression test, fix bugs, regression test and so on.

Hemant Elhence: Jim, let just take one quick question. Did you have any resistance from developers to embrace TDD?

Jim Smith: We did. Yes, sir. Yes. Yes because there were a lot of developers that weren't doing TDD previously. The next click towards laxness from TDD is just plain old doing unit test. We had a lot of developers that did unit test but the whole idea of writing them before you write your code was pretty foreign to a lot of folks. In fact, when we first kicked the product off, we started that without a very brief commitment to TDD. These days we're very committed to it. The result is improved quality, one of the results of it.

Hemant Elhence: Okay, thanks.

Jim Smith: Sure. Lesson seven; in all levels, current pushes back towards waterfall. You're going to hear people as they start to do Agile at the same time that they're experiencing great things from Agile, they're going to have their complaints. There's still going to be some issues. They're going to say, "We need more documentation." That the current level of lean documentation that we're producing is not enough. You're going to hear developers saying, "Hey, this is great. Thank you for allowing me to think for myself but, can you do a little more thinking for me?" You'll hear developers and QA testers say that sort of thing.

You'll hear some customers are going to want more documentation out of what you're producing. You're also going to hear, "Hey, we need more time up front to figure this stuff out before we just charge into it," because certainly when you just charge into it, sometimes the team makes bad decision . Of course when you're doing your retrospective and looking back, one of the things you say is, "IF we didn't just charge into this, if we figured some of this out upfront, then we wouldn't have made that mistake." Remember the old waterfall saying that a little bit of design can reduce a lot of programming, bug fixing or something along those lines? You're going to hear people asking for these things.

More time for regression testing. We need more time at the end of the sprint. Then, you're going to see more email, IM, and bug record correspondence too as people talk to each other face-to-face less and less often. All these things if they start becoming really strong are going to push you right back towards doing waterfall. You're going to end up with this funny form of scrum bug that involved basically having these

"iterations" but iterations really consist of all the different phases of the waterfall development.

You have to be mindful of that, and for some of these things you have to resist. For some of them, you got to feed them a little bit. For example, when you hear from a developer that they need more documentation, you got to ask the question, "Do they really need more documentation?" and you got to talk it over. This is part of the reason why it never ends. The process never stops changing.

Lesson number eight. Even the best and brightest have trouble with collaboration. So when you talk about scrum or Agile development in general and you're talking about people collaborating, breaking down those silo walls, it's a very exciting topic. It's really the way everybody wants to do things.

These days developers have to be social whereas in the old days it is acceptable for developers to be the kind of people they just want to be talked away in the basement office just churning out some code every now and then. It's a very exciting idea to imagine everyone collaborating. The idea of community, the development community. The experience is it's real hard to get even your best folks sometimes to collaborate on things. Some of the same issues remain.

For example, if I give a developer a piece of a library and I say, "Hey, let's use this instead of reinventing the wheel." The developer is going to have a find a little bit of urge to rewrite that library, to read, to just write the code themselves rather than learn what's already in place. You're going to have similar issues with other groups of folks.

Get them to talk to each other and then collaborate. Mostly, it's because, I think, they're very easy and they're staying focused on what their specific job title entails but you got to get them to collaborate. They've got to talk. You got to do all that you can to get them to talk to each other but it's hard to do that. That's just lesson number eight that you learned even though it's a required skill.

Lesson number nine. The team will gladly turn in slackers. This is one thing that really … This is almost like one of those seen the pyramids things for me too. I was amazed at how previously if you had a developer that wasn't pulling their weight on the team, the team would just sort of tolerate them.

However, once we switch over to Agile development and you start having these deadlines, basically the end of experience, the day that you're going to have your demo, the team is glad to say, "Guess what? Larry here, the developer, is not pulling his weight. He never comes to work. He starts working on something but then, these other guys have to finish his work for him." You'd be amazed at how the team is glad to turn in those folks to you and to say, "Hey, do us a favor and get him off of our team."

Anyway, it's a help. It's an aid when you're a manager. It means you don't have to go around micromanaging people because the team is glad to turn in those folks that are not pulling their weight.

Hemant Elhence: A quick question on this topic.

Jim Smith: Sure.

Hemant Elhence: Just colocation of teams. Do you have collocated spaces for your teams to allow visibility affect us and so on? How does collocation fit with the other?

Jim Smith: Great question. We didn't have beliefs of cubicles. However, at the same time, we allow our developers to work three days a week from home. Just using simple collaboration tools like Link, for example, Microsoft Link to detect someone's presence and you're going to detect their presence when you send them an IM and you get a response or don't get a response, or when you call them and get a response or don't get a response.

That's another lesson that we've learned. It's not even in this presentation, it's that you don't have to have everybody in the same little cubed area for everything to work properly. We do recommend that they're on the same continent, the same time zone on a given team but you don't have to have everybody sitting right next to each other. If you look at the layout of our offices, you'll see that we have these little base, for team one, and you'll have a group of cubicles.

We had great expectations in mind for that scene. We thought, "Hey, having everybody sitting right close to each other is really going to do a lot to spur on collaboration." We found it really didn't help that much. What really helped was giving them collaboration tools like Link, for example, and enforcing on them that we expect them to be talking to each other often.

If they come to you and say, "Hey, I haven't figured this out," and the answer lies with another developer, you just ask the question, "Have you spoken to him about it yet?" In time, that just becomes the norm and people just start going to those other people and expecting them to be available." I probably gave information than you needed for the answer to that question.

Hemant Elhence: No. I think this is good. This is very good. Just keep going. I know you have more lessons.

Jim Smith: Okay, great. Yeah, we still have a bunch to get through. I'll go quickly. I'll speed it up a little bit. Lean documentation is still good. Documentation is still necessary. A lot of things think, "Hey, once I make the jump over to Agile development, we're going to stop producing documentation," and some people even have fear about that but documentation is still necessary.

You just have to figure out the amount of documentation that is necessary. You know how it is. Too much documentation and it becomes obsolete quickly, or it just claimed wrong, or if it costs a lot of money to maintain. You want to find that nice balance between documentation. It is useful while not at the same time being too expensive to maintain and it is keep up-to-date.

Lesson 11, "Nothing is over. Nothing." This is a quote from John Rambo. In other words, as I mentioned earlier, your approach is all going to evolve. Do as prompt and then give the team the opportunity to have retrospective, and then the process falls on the retrospective, and make modifications. That's going to continue as far as we can tell until the end of time because we've been doing Agile for ten years now and we're continuing to make tweaks.

Lesson number 12, the team needs to understand that you can't do everything that falls out of retrospectives. When we first started doing retrospectives in earnest, one of the pieces of feedback we got from mark, our staff members were, "Hey, I still need to do this in our retrospective. It both appears to be moving to get that done."

If you're a manager and you're on the line, you know that you can't implement everything that everybody wants to do. We recommend that you tell folks right from the beginning, these retrospectives are for the purpose of helping us improve the way the team is doing things. However you can't expect that we're going to be able to do all of them.

Please have that little maturity, folks. You're going to have to tell them that. They're also going to be unhappy.

Lesson number 13, although it's not ideal, team members can be scrum masters. It is possible. You heard debate sometimes that you have to have dedicated strum masters for the team and then you hear sometimes that you're just going to end up with a mess if individual team members aren't strum masters. We found that's not the case with PDX. It is possible for a person to make maybe 50% if their time to allocate it towards the duties of strum master.

It is certainly better when you get individuals whose only job is to be a strum master. They can just have more time available to do the things that a strum master is supposed to do, but sometimes you have the budget to do it and it is good, it's useful to have individual team members do it sometimes. In fact, we sometimes rotate that position among team members. It gives the team members another perspective on things.

Lesson number 14, you can get executive, managerial, and customer buy-in with your first demo and through training on users. That's the key. Having a demo and having the demo be more than just a market has great effects. It rebuilds a lot of confidence with your customers and the rest of the folks within your organization that you're really getting something done.

As I mentioned, when we started working on EPS, for the first several months, our projects looks like parts on the garage floor. We couldn't really demonstrate a whole lot and what we could demonstrate was just a markup but once we started going Agile, we started out with something we can demonstrate nicely. Every button works. Nothing will stop the functionality available just increasingly became more available. We started having more and more functionality in time.

Having those demos does a lot to get them not only to have confidence but they have the buy-in to say, "Hey, these guys are really getting the job done. We got to give them the support that they need to get it done."

Lesson number 15, having customers and other stakeholders at demos is a good thing. One of the whole points of agile is that the end of an iteration, you got somebody verifying for you that you're on the mark. That you're not producing something that they didn't ask for. Having them at demos is a good thing

Now, it seems some organizations have said, "There's no way we can get our people to show up for this sense. No way we can get our customers to show up. They're just too busy. They don't think this is important." I recommend that you go back at them again. You do all that you can to get them to show for these things because it will save a lot in the long run. Also, incrementally, introduce them to the product and they'll start getting more and more excited about it as more and more functionality become available.

Lesson number 16, let the scrum team stay focus; retain a production support team. One thing that we found early on is a textbook problem with Agile development jobs in a lot of cases. The whole point of scrum is you got lots of folks concentrated on solving a problem but what happens, reality is "Hey, this one system is down. We need Larry to stop what he's doing and to come over and help us get it on the weekend." That can have a bad effect on the team meeting their deadline.

What we did at PDX, we're not the only team who do this but we have a production support team, a group of developers whose job it is basically to keep the production up and going. If there's a problem that requires a developer's expertise, the support folks that our company draw on other people, and those people who are also turning out bug fixes that are critical if they find any in production. That enables the scrum team to stay focused on what they're doing. You can rotate membership on that production support team.

One thing that really has been amazing to me is that some folks, you're going to find actually enjoy being on that production support team and from a high level, when you hear the words production support, that just sounds like the anti-glamour organization but really, there are folks that enjoy the heck out of working on those things because there's always a lot of action.

Lesson number 17, get your DBA team to agree to an SLA. Often times, DBA teams are left on the equation when you're talking about how does Agile work. Often times, it's because the DBA team is basically keeping databases up and going and they usually have their own waterfall-like process for churning out column changes, tables changes, human changes in general for your databases.

One thing that I recommend that you do is you draw your DBA team into this whole thing and make sure that they are abiding by SLAs. You don't want developers to say, "We need this new column to implement this new functionality but we've got to wait six weeks for a DBA to make the

change to schemas." You don't want that happening. There are some organizations where developers actually make scheme changes. Work on one of those shops and I know there are a lot of companies out there that are thought like that that is well thought off.

Lesson number 18, break down those user stories. Sometimes user stories turn into very big areas. They can be very general. They can really encapsulate a full new module and some let alone, one or two operations that are necessary. Really, you basically just breakdown user stories. If user story has a lot of great points associated with it, and you can say, "Hey, man. It needs breaking down," and turn it into something that's a lot more digestible.

Lesson number 19, keep noisy managers and executives out of kickoffs. How can I say this? Someone told me and then other people confirmed that, "I think Jim Smith needed to stay out of kickoff meetings," and it's because I just happened to be naturally the kind of guy that wants to just tell people what to do or to call baloney on people saying, "That shouldn't be so hard," or "Hey, we're going to have the statistics." It's a bad practice to have guys like Jim Smith in kickoff meetings being allowed to talk a whole lot.

One good idea is to just keep those really noisy people out of the kickoff meetings. There's still a way to get their input but it's certainly not to expose the team and for the team to start feeling the pressure of those folks. Ideally, your team members would push back on noisy executives and managers. If the executive has a funnel of senior vice president, this, or that, or director of this thing or that thing, then a lot of them, just by nature, turns out that a lot of development folks are not confrontational. They don't want to stand up to this and tell them, "Hey, I am sorry. I disagree with you on that." A sense of just, being there. If you like your product one to be good and you keep those noisy guys out of the room.

Lesson number 20, than stand up is for everybody, not just for the scrum master or for a project manager to express what's on their mind or to get what the team is doing. The team wants to be listening to each other. If Larry, the developer, for example talks about what he worked on yesterday; if Mo, the developer, if a flag builds up for him, then he should know it. Then, after the meeting, they should talk about it. They should go offline with those. You got to be careful because daily meetings sometimes devolve in these ugly things where everybody is just standing there giving their status and there's not really any kind of discussion that happens afterwards.

When really the purpose of that thing is so that everyone can communicate with everyone else. It's a way to insure that at least, 15 minutes every day, the team is gathering and talking to each other. You got to make them talk. Don't let it devolve into just a status filling session.

Lesson 21, the product owner has to appreciate the value of paying technical debt. Usually product owners are folks that understand the business. That's good. They ought to understand the business but they also need to understand the importance of paying technical debt. I sometimes hear people talking about that being a challenge is just insurmountable, and I have to call baloney on that one because if someone is bright enough business-wise to be product owner, they are certainly capable. They have the intellectual capacity to understand the importance of paying technical debt.

If, for example, you're on version 1.4 of Java and the product owner doesn't have the appreciation for the dangers of being on that ancient version of Java, then you got a bad product owner on your hands. It is possible for the team to say, "Hey, look. You need to understand why it's important to pay this technical debt. You could have security problems. High profile security problems are going to be bad things to your product and your customers, just as an example.

Hemant, I think we did the 12:45. Yeah, we did hit 12:45. I am only on lesson number 22. Shall I continue? Just go on ahead?

Hemant Elhence: Yeah, continue but just take pick up pace, Jim. I think there's value in these lessons so we'll squeeze the Q&A time, we asked the some questions along the way. We'll just pick up pace only.

Jim Smith: Okay, great. I'll try and move faster. Product owner is not the team owner, lesson number 22. In time, the product owner because they got so much on their mind, they got so much input, specifically they got so much pressure coming from customers, executives, and so many other third party vendors that they often times take it out on the team, and they say, "Hey, you are going to do this and you're going to like it. You're going to do that." That's generally a bad scene. You want the product owner to be dictating the priority of user stories but you don't want them necessarily to be bossing around the individual team members.

I can go on and on about that but I need to dump to number 23. The people arguing each other, people slamming doors and slamming doors, I don't mean literally slamming doors. I mean, they're just not helping

each other out or they're putting up walls, then you got a dysfunctional Agile team and you need to address the problem or else, you're not going to get continually better.

Lesson 24, parties and other rewards after demos equal good. I am going to stop for a moment on this one and say, this is really an important one. A lot of development managers don't have an appreciation for the importance of this one. A lot development managers got to where they are by working hard and work is just itself something that they think is something that is worth doing and they don't even need a reward in a lot of case basis.

You've got to keep in mind however that your foot soldier can appreciate some type of reward. A team had a good demo, take the team out, not to lunch. Have a happy hour event. Take them bowling, do something fun with them. Do something to just make a big deal on the fact that they just succeeded because they won and the deserved it. They are the ones that are working really hard in getting the job done and you owe it to them to give them some recognition.

It's less about the free lunch, the taste of the free lunch or the satisfied feeling of something in themselves and more about the recognition. You go to make sure you're giving that and it's a very easy thing for most development managers to neglect. Don't neglect it. It's an easy thing. Just do it.

Lesson 25, don't let the sprint go longer than five weeks. If it goes longer than five weeks, then basically, you're going to wear the team out but I am going to rush ahead a little bit.

Lesson 26, train developers to sign off on user stories early. Some developers want to wait until the very last day, until Q&A is done, no more problems with the software before they say, "Okay, I am going to sign off on this." The bad effect that's going to have is it's going to make your burn down charts look ugly and you're not really going to be sure how far along when someone says, "How is it actually doing?' I guess, the trick is to start now and just do those early.

The product owner position is a full-time job. This is an important one, lesson number 27. Some organizations, when they adapt Agile, they're out there looking for a product owner and they're like, "Hey, let's get a vice president of sales to do this job." In my experience, that never works. We have not really done that thing in PDX thankfully much. We did a little bit but I've seen other organizations doing it and then saying at

the end, "Our product owner really doesn't exist because he's out on the road all the time," or "He doesn't even maintain a backlog really. He has no idea what we're working on." Your choice of product owner really ought to be someone that's dedicated to the cost. Not someone that has another fulltime job.

Lesson number 28, we already talked about this one so I am not going to talk about it much now but colocation is good but it's not absolutely necessary.

Lesson 29, your development and test environments are production environments. You got to make sure that they stay up. This one is independent of Agile but I notice that when you start doing Agile development, if a development environment is down, it really, really introduces a lot of anxiety. Just imagine your developer and you committed to getting the job done by a certain day, and then all of a sudden, your system is down. You can't do any developments. That's a bad thing.

You got to make sure that your IT team understands that you got commitments to fulfill and they too have to be Agile and get your systems fixed for them. You want to make the investment in your systems too to give them all the redundancy that they need to be able to stay up because it will be worth it. If you just set in terms of developers being down with nothing to do or having to find something of questionable usefulness to do when their systems are down, then you can imagine the waste involved.

Lesson 30, we're on the homestretch here. Don't treat your India folks like warm bodies or they'll act like warm bodies. Before we put our current India team employees, before we built a team in India using Synerzip by the way, before that, we had two attempts at India outsourcing and they didn't work very well. I could talk easily a few days on this topic but I'll just say if you think that when you got an Agile team that your India team is just going to be a black box, basically a bunch of nameless folks churning up codes, then you're setting yourself up for failure.

A fundamental part of Agile development is that your team members are plugged in to each other, they're talking to each other. Part of that has to be that your India team or your offshore team, wherever they happen to be, offshore or near-shore team, that you're plugged into them, you know their names, you know their faces, you know a few things more about them, and that you're not just considering them to be a black box

churning out code. Anyway, if you do that, then they're going to act like a bunch of warm bodies and you'll have lots of turnover and they won't get the job done the way you want it done. Right from the start, never think of an India team, or your offshore, or near-shore teams being just a black box.

Lesson 31, Sashimi is a great idea but not always 100% possible. Most of you that don't want a Sashimi thing is it's the concept that you divide your work vertically. In the old world, you had developers that were UMI developers and backend developers and middleware developers, and each of that worked on their particular layers.

The Agile way is to divide your work vertically. If you got a new piece of functionality, then the same developer might work on the user interface, the backend, and the middleware and so on. They'll do all of it and they'll get the job done. One thing you, just have to face those and you'll learn is that is not always 100% possible because sometimes you got developers that are junior level and they're not capable at all at working on all layers with your stack. That's a lesson you learn, but you should try for it as much as possible. It will make your life easier.

Lesson 32, velocity steadily increases when you team rosters are constant and working on same products. If you often change your team rosters, then you're not going to get up really enough for your velocity to increase that way. We're almost done here.

Lesson 33, the product ought to be ready for production after every sprint. Even though it's not always possible, you should try and make it happen for every sprint. If you're finishing up a sprint and some of your code is just mopped, your user interface is just mopped up or part of your code just plainly don't work if you take it off the road and it starts running all kinds of exceptions and error, then you're not really getting the job done and you're basically just weaving a lot of loose ends for future cleaning sprints.

If you start configuring into that early, if you start early on accepting that it's okay for your code to not be production ready when you're done with sprint, you're going to end up with big ugly ball of broken code at the end. You want to make sure that at the end of every sprint, it is production ready or at least, it's close as possible to be production ready.

The scrum master is a strong servant leader. He's not George Patton. He's a guy that basically is just saying, "Okay, the team needs to get the job done and I am going to get rid of whatever impediments they identify

to them getting the job done and that's what I am going to do. If your scrum masters are out there running the show instead, that's a bad sign.

Lesson number 35, plaster that product backlog everywhere. Just keep to everyone on the same page at all levels, make sure your company's brass knows it, knows where they can find it. Make sure that your customers know it. Get it out there because the danger of everyone not knowing about your backlog is that you might end up working on something that's not important or you might fail the work on something that's critical that has to get done right away.

When you're plastering it everywhere, putting it on web pages, putting it up on the wall, forcing it down people's throats and meetings where you review the backlog and so on, sending it through email messages to people, then you can keep everyone on the same page, but it does require a lot of work. You can't expect, for example, for an executive, a CEO, let's say, to log into your backlog and to see your work, the top priorities on a human product backlog. You got to take it to them and say, "How's this look?" There are a lot of techniques to doing that but we're almost out of time so I can finish here.

Developers are professionals. They're not privates in the army. Treat them as such. Developers, whether they're programmers, QA testers, designers, DAs, architects, they all have some education and they all worked hard to perfect their craft, and you cannot treat them just as you would treat a private in the army, let's say. You got to treat them as professionals. You got to respect the work that they do and you got to keep feeding them good stuff including a constant supply of continuing education and you got to respect what they say to you.

If you try and downgrade them to privates in the army, then they're going to behave as privates in the army and they're going to do exactly what you tell them. When you do that sort of thing, you are forfeiting. For a developer for him to do that, you're forfeiting the power of their brain. It's a bad thing to do.

This is the last one, lesson number 37. Scrum is not an acronym. Don't capitalize it. Don't put little dots after each letter in it. It refers to that very ugly – sorry, that's abusive to hear but that ugly formation in rugby where everybody is concentrating on the same goal at the same time.

That's it. Are there any questions? Do we have any time for questions?

Hemant Elhence: Yeah. I still can squeeze a couple of questions. If you can just advance to next couple of pages. We already have a bunch of questions. Let me do a quick overview of Synerzip just in 30 seconds, if you can get to the next page, Jim.

For all the audience, people in the audience who don't know Synerzip, we are a software product development partner for companies like PDX, in this case. We help plan to adapt Agile and then do good ongoing Agile development work with dedicated team for each client just like the story you hold with PDX.

Let us know if we can help. Let's move to really following some content and questions. One question, Jim, that came up which is a good one to consider. It's about, how much not to do upfront? Can you talk about how much architecture and software design to do upfront is a part of good Agile practice?

Jim Smith: That's a tough one. It's objective and it's something that I think we constantly adjust to. In fact, PDX has adjusted several times. I'll just give you an example of one little snapshot in time because no particular scheme basically addresses every situation. I'll just tell you like at one point, basically the way PDX was doing it was we would have a preplanning period of about two weeks before a series of three sprints.

That was an opportunity about quarterly or so for everybody to get together. We would bring in all kinds of folks in this meeting that would last for about two weeks to review all of the stuff that is coming up. Architects, we'd we get the opportunity to lay up the architecture, DBAs, and get the opportunity to lay out the database and the design team have the opportunity to design the user interface and so on.

Then, we jump into each individual sprint or we jump into the first sprint and tweaks were possible between the first, second, and third sprint but basically you would, in the end, have all these slight tweaks in sprint two or sprint three after that initial preplanning, but we modify that quite a bit throughout our history. There's no right answer I think.

Hemant Elhence: Let's take one final question, metrics. What metrics you guys follow now that you're way along Agile adaption journey and how are those metrics different than what you had before you started Agile?

Jim Smith: A lot of our metrics are based around user stories and story points. You have to be careful when it comes to story points. Story points, they're like currency. Just because one team is assigning a higher number of

story points to a given piece of functionality than another team, it doesn't mean it's necessarily significantly more worthy or more important at all necessarily, but within a team, those metrics can become useful.

There are lots of different kinds of metrics. There are quality metrics, there are velocity metrics and so on, but a great example of the quality metric is the number of defects per story point. You always want to have great velocity for a team and you want great quality and a lot of folks argue and there's truth in this. A lot of attention to one means basically neglecting another. In other words, if you start putting a lot of effort into getting velocity, then your quality might start to suffer.

One way that we found to balance that is to use basically the number of defects for a sprint. That's a good way, we think but again, it's only meaningful per team.

I will say one other thing really quickly. This is well known in the Agile community but a lot of folks adapting Agile don't get it. You really have to care about using story points to gauge velocity. It's really hard to measure the velocity with team and to basically hold them accountable for maintaining a velocity. They say, "Hey, you normally get 20 story points done per sprint. We're going to hold you to that every sprint." You can just think through it and imagine some of the pain you could introduce by doing that kind of thing. Developers are just going to start modifying the number of story points that they assign. Very difficult to do that.

It's also very difficult to measure velocity performance per team member. It's easier to do it per team and again, that's a pretty well-known one within the Agile community but a lot of development managers either don't know it or they ignore it and they pursue that one anyway.

Hemant Elhence: That's right. I think that also that could be a topic by itself on Agile.

Jim Smith: Yes, sir. I've attended many of those, you'll see them on a regular basis.

Hemant Elhence: Absolutely. Thank you so much, Jim, for sharing. I know there are many more questions. I encourage people who attended the session today, if there are any of these questions you would like to follow up, let me know. We'll figure out a way to do it. Thank you all for attending and thanks again, Jim, for being fast to go with these lessons. It was excellent content.

Jim Smith: Thank you, Hemant. Have a great one everybody.