Role of Software Architect in Agile Projects

written by Manoj Khanna | January 2, 2017



One of the biggest misconceptions about Agile is that architecture is not required in the Agile development. ‘We‘re Agile; we don’t need architecture’—is something that everybody involved in Agile has heard at least once.

Let’s start by establishing a common understanding of what the software architecture is, to which everyone can agree. The definition of architecture is quite broad, and the roles and responsibilities of software architects vary dramatically from company to company.

Here is how Martin Fowler identifies architecture in Patterns of Enterprise Application Architecture:

“Architecture” is a term that lots of people try to define, with little agreement. There are two common elements: one is the highest-level breakdown of a system into its parts; the other—decisions that are hard to change.”

I find that we all can agree on those two common elements. Do we need the highest-level system break down? Absolutely. Do we need huge documents and long design stages? No. Agile is not against the architecture—it’s against useless, bulky documentation that nobody reads anyway.

From the perspective of change, the role of architecture in Agile development becomes quite clear – A good architecture is responsive and will support agility; a poor architecture will resist it and reduce it. And, since one of the benefits of adopting Agile is a better response to changes in the requirements, it's obvious that flexible and extendable architecture is a key to this.

The biggest issue that I've noticed is the very thin line between architectural design and software design. I've seen companies where the different implementations of the following practice were used: Architects created design documentation and developers were responsible for writing the code. This introduced a myriad of problems, starting with developers feeling that they were not fully trusted. This also gave developers an excuse not to really think about the design. 'We're just coders, not responsible for the design and we do only what we are told to do.' is a common attitude that I have witnessed. In Agile, the developer is responsible for the code he writes (and unit tests) as well as the design since nobody else will provide him with it.

Ideally, the high-level software architecture is completed before coding starts. And I really have to be careful here – completed doesn't mean written in stone; it can change, but with an understanding "this is the best of what we know right now." This doesn't necessarily include a database design or class diagram, and the level of details really depends on the approach you will be taking moving forward. I found that for certain systems the Domain Drive Design (DDD) is extremely useful and has made my life far easier. Therefore, I like to have a domain model and a basic set of domain classes and their relations defined, but not to the level of methods and attributes.

Personally, I prefer projects to have a design stage; this is when the high-level business domain model and user stories are created. At this stage the main architectural decisions are

made – the technology that will be used, the database server, the application type (for example, Mobile, rich client, or service), the architecture style (client server, layered architecture, SOA) is selected, and the architectural frame that will be applied is selected as well. The document created during this phase is not solely architectural effort, it is a collaborative effort of business analysts, developers, and network administrators. The output of this design stage is not only a high-level architectural document. (This is not an attempt to make fixed predictions of the future or create a detailed software design upfront as this approach places all the significant decisions at the point of least knowledge in a project's lifecycle). This is simply a way of getting and sharing the common understanding of the system we are all about to develop.

This is an excerpt from the forthcoming book, The Art of Being Agile.

[Image Courtesy: Flickr/Helix/Philip Gunkel]

Rise of the Chatbots!

written by Manoj Khanna | January 2, 2017



In light of building a contemporary digital experience and social engagement, the rise of the Chatbots is quite an advent

when combined with the latest tools & technologies. We have clearly seen a growth in digital concierge services from Apple (Siri), Google (GoogleNow), and Amazon (Echo) in past coupled of years if not more – the use of common language and communication with digital devices is increasingly becoming a standard. As Dion Hinchcliffe explicitly references, IRC Bots, Eliza, & Zork – the latter, command line programs from 80's, and the former Internet Relay Chat (IRC) that used to perform automated functions to control the IRC Channels back in the day when I was in high school working on dial-ups.
Today, the world is different, and the Chatbots are a self-learning and evolving machine – they are the new frontier for brand & consumer interaction. Uber's Messina describes it as 'Conversational Commerce', and Facebook's Zuckerburg describes as the 'next big thing' that is being worked on now at Facebook, and they've built a Chatbot roadmap of sorts.

How do Chatbots work?

Chatbots work differently, and they have a human-built intellect that is fed over a course of time and developed using data which is curated based on an archival process of scenarios and cases. The knowledge experience part comes from the business logic and machine learning, and the constant communication of the connected devices (apps, devices, APIs, DBs) which are feeding into the business logic. So your Chatbot essentially becomes a self-learning machine which should get better and better over time. The information feeders to Chatbot are multi-channel user interfaces – so any data that is visible to us eventually gets fed into its knowledge portal. The accumulated data gets further curated through machine learning and is then queried on through algorithms which utilize the power of cloud computing.

So what's in it for companies?

For Social Media companies – it's about connecting users with their brands, and for brands, it's about their products, brand loyalty, and customer service. And this all leads to the monies. Companies, or in this case the guinea-pig pioneers within social media and brands that are looking at Chatbots as

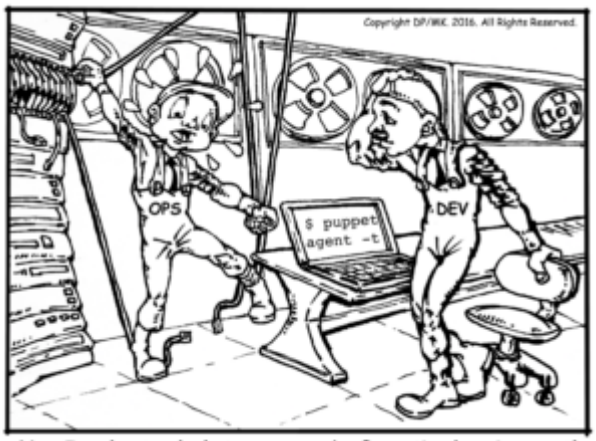
a way to monetize into the building hype of ‘conversational commerce’, and also, as a way forward to potentially change they interact with customers today.

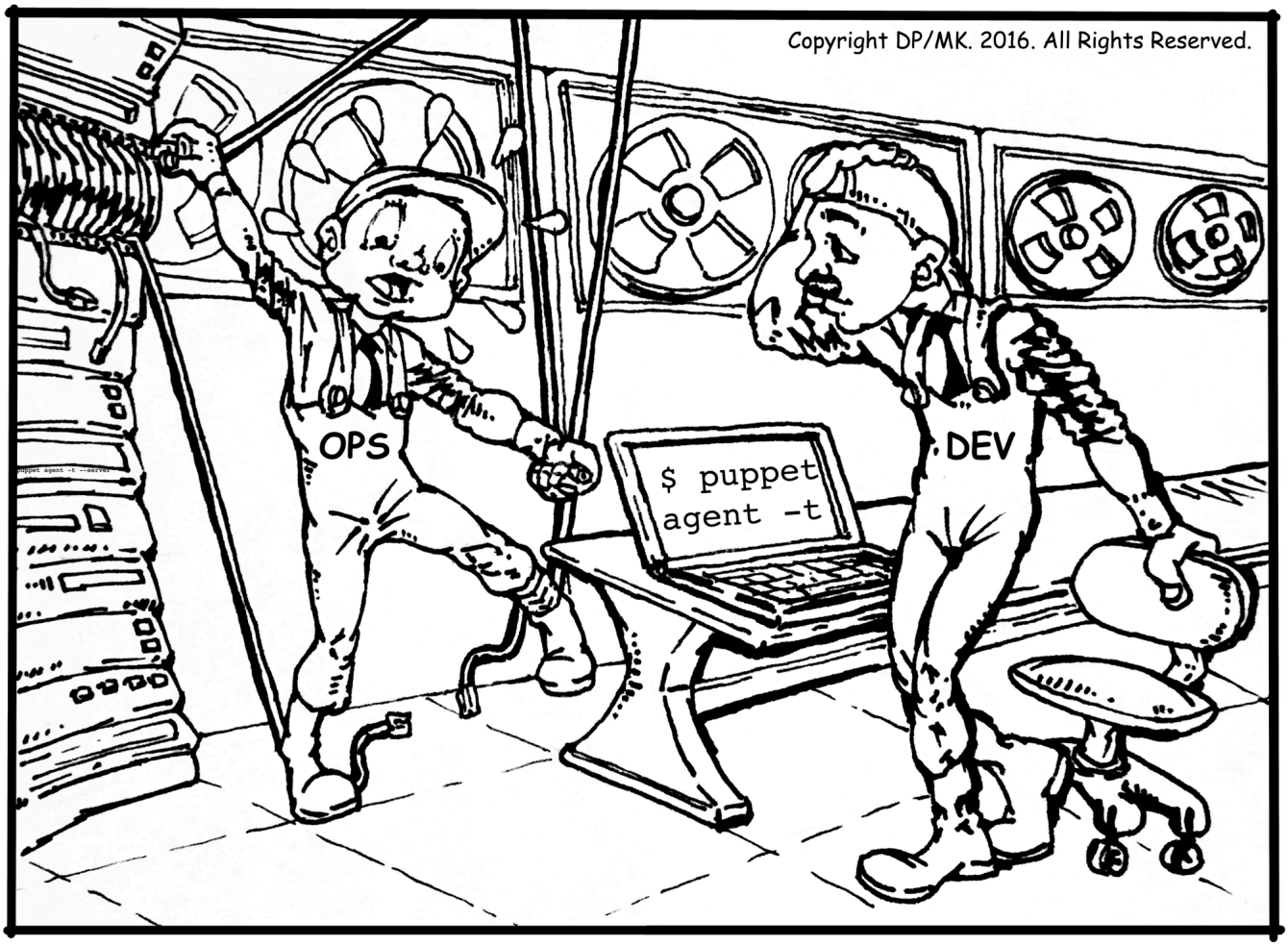
So how should one get started on Chatbot?

Follow a minimalistic approach – solve a simple problem and then bring complexity. The power of natural language – where users can query simple things – such as service type, any recommendation or what next product to choose from – could be a simple but good start. Worth for brands which are user conscious.
[image courtesy: tabletop assistant / MattHurst via Flickr CC Licence By]

Puppetized!

written by Manoj Khanna | January 2, 2017

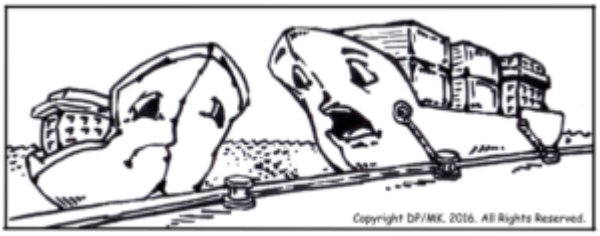




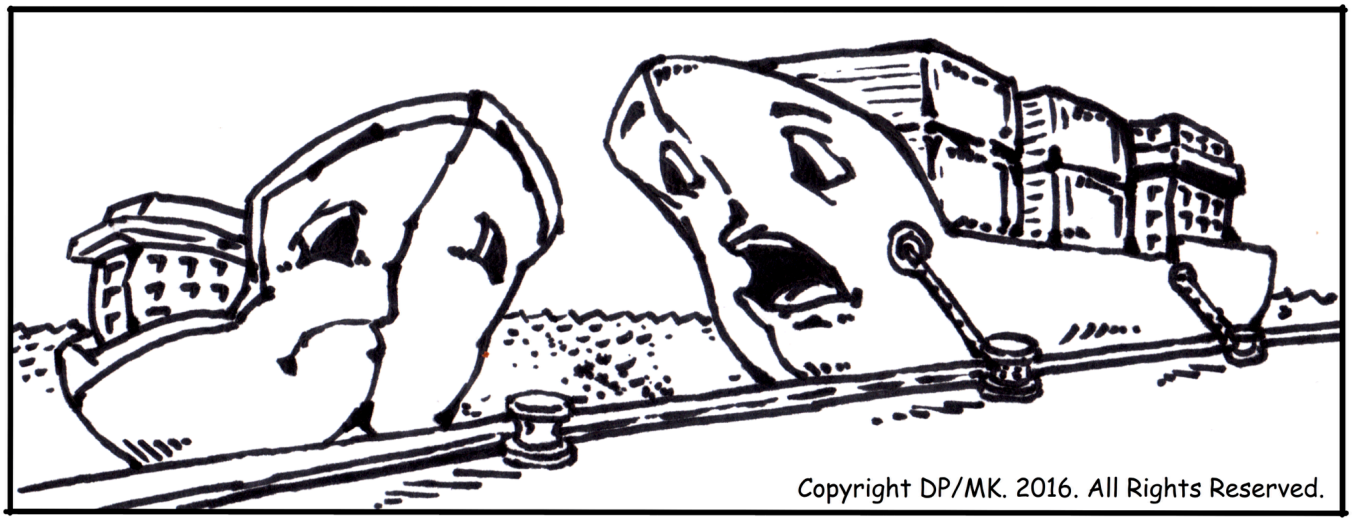
Now I understand what you mean by Puppetized environment!

Dockerized!

written by Manoj Khanna | January 2, 2017



Get Dockerized? You're not even Containerized, yet!



Get Dockerized? You're not even Containerized, yet!

Understanding the feedback in 'The Feedback Loop'

written by Manoj Khanna | January 2, 2017



What is Feedback?

Feedback occurs when the return of information concerning the results of a process or activity takes place (http://www.thefreedictionary.com.). This event is part of a chain of cause-and-effect that forms a loop onto itself. Feedback comes in two forms: good feedback and bad feedback. Without feedback, the Agile process of inspection and adaption

could not occur. An Agile process thrives in an environment with constant change. Because of this variability, adaptation and adjustment points are made on a regular basis. If we can shorten the amount of time elapsed between these calibration points, then we can more quickly adapt to these changing realities. In short, this is the feedback loop in our process and environment.

Types of Feedback

There are, however, different forms of feedback, which are listed here for the reader's reference and throughout:

1. **Communication feedback** (e.g., onsite customer, open team workspace, ubiquitous language)
2. **Technology feedback** (tools we use to give us quick feedback, like Cruise Control, mocking, BigVisibleCruise, CCtray, Resharper, TFS real time compilation/warnings, and confirming that we use the right technology)
3. **Requirement feedback** (when a customer need realizes in a demo or the production environment)
4. **Market feedback** (to see how the market reacts to a new story/feature/module, frequent and numerous deployments)
5. **Analysis, design and coding feedback** (e.g., pair programming, whiteboard mockups, code reviews)
6. **Defect and testing feedback** (the quicker we find out about bugs, the quicker we can fix them, deploying often and always, test driven development)
7. **Operational feedback** (process, methodologies, practices and how to improve them)

The quicker we can get these forms of feedback from the source, the faster we can validate our progress and adapt to the information received.

> *A company's ability to deal with change and adapt accordingly to changing conditions will improve its competitiveness in*

> *the marketplace. Companies that struggle with a slow feedback loop will find themselves caught up in trying to solve problems that have already changed or are not important anymore.*

Effects of a fast feedback loop

Having a fast feedback loop allows dominance of a company during market changes. For example, one of our clients using our health and safety management system had a deadline for submission of reports. Approximately 800 companies assigned to each performed various tasks and submitted reports showing the work completed. There were some features and latency problems that the client wanted to be fixed, and the deadline was one week away. We were able to get quickly the high-priority features added, and latency issues resolved three days into the week using our engineering practices, automated testing, and automated deployments. We went live before the deadline to get much-needed feedback on our changes in a real production environment. If we had waited until after the deadline, we wouldn't have obtained the actual feedback from the end users re the added features, nor the feedback from an environment production perspective, since the client wouldn't be using the system until the next deadline, which was months away. Using this feedback from deployed features in a production environment allowed us to make more improvements so that the next period time would go even more smoothly.

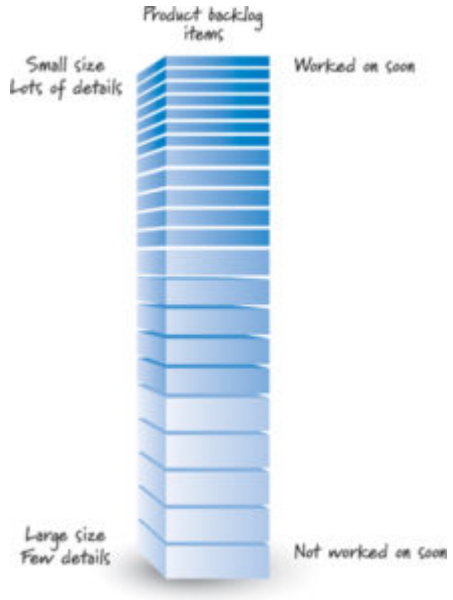
Summary

Being able to perform a full cycle of development from client request to production deployment in a few days helps ensure the company can quickly adapt to changing market conditions.

(This is an excerpt from the mini book series "Agile from the Trenches: The Feedback Loop")

The Significance of Product Backlog Refinement in Scrum Success

written by Manoj Khanna | January 2, 2017



Product backlog refinement, or PBR, is an integral component of successful Scrums. This process of continuously reviewing product backlog items, to ensure that teams know exactly what to work on in the sprints, cannot be done without. It keeps the teams and the product owner on track. To understand why PBR is critical, it is necessary to first understand the details of what PBR is, what we expect from it, and the strategic value it provides.

What is PBR?

Product backlog refinement is the process in which the product owner, ScrumMaster, and the development teams review the product backlog items and define the stories that they will need to work on during the immediate sprint. Epics or unclear

stories are split apart into smaller stories and are detailed in this process, while unnecessary backlog items are removed. In so doing, the backlog items are analyzed in terms of how much time and work each one will require, and the requirements for each item are clarified. After the PBR session is complete, each story should be valuable and testable.

PBR is conducted in each sprint planning meeting to address the tasks for the immediate sprint. Thus, for each project, PBR is conducted at least as many times as the number of sprints. New Scrum teams, or those with sprints of more than two weeks, may find it useful to conduct more PBR meetings than the number of sprints in their project; additional meetings would be conducted outside of the sprint planning meetings, and these could compose up to ten percent of the teams' time. The frequency with which PBR should be conducted is due to the volatile nature of Scrum product management. The completion of each sprint reveals more details regarding the product, which results in the need to alter stories – by adding or deleting certain aspects of the stories – and update.

Expectations from PBR

The process of PBR is conducted with the intention of thoroughly prepping development team members about the sprint's tasks, such that the teams know precisely what to work on and achieve by the end of the sprint. It also is conducted to assist the PO in getting the top-priority backlog items ready for the sprint planning meeting. Essentially, product backlog refinement occurs with the purpose of clarifying each sprint's tasks and ensuring that they are in sizable chunks that can be accomplished.

Each PBR session is intended to provide team members and the PO with opportunities to update their sprint tasks accordingly, while additional PBR sessions outside of sprint planning meetings enable them to work on detailing and

refining a larger number of backlog items.

Vision and strategic value of PBR

Product backlog refinement assists in ensuring progress toward the project’s objectives. The process succeeds in keeping the PO and development teams on track toward the project’s main objectives, as it is a way for the PO, ScrumMaster, and development teams to maintain a clear vision of the sprint tasks, especially in light of product feedback, the emergence of new ideas, and changes in project needs – aspects that can occur throughout each sprint. Scrum product management involves constant shaping of the product, which necessitates redefining the backlog tasks in each sprint. PBR optimizes such redefining of backlog tasks, thereby preventing the PO and teams from losing sight of the work to be completed in each sprint. In so doing, PBR keeps the PO and teams on track during the project.

Not only does PBR keep the PO and development teams on track but it also provides strategic value in that conducting PBR prevents a number of issues from arising later on in the sprint. A main issue that PBR can help the PO and teams to avoid is delivery of stories that do not meet the recipient’s full needs. Without PBR, teams work on stories whose details are not clarified, leading the teams to complete the stories unaware of certain requirements. This results in a loss of time and effort. Another related issue that PBR can make avoidable is a volatile team velocity that results from running ambiguous sprints. If a PO and the development teams choose to forgo PBR, they also risk completing sprints where the stories they worked on were not the most valuable stories. Had a PBR meeting been conducted, the more valuable stories could have been noted and addressed accordingly.

Summary

The value and benefits associated with product backlog refinement are large, as are the consequences of forgoing the process. On top of the value it holds, PBR can be easily integrated into each sprint. Therefore, there is little reason to pass over this process that is critical to Scrum success.

[This article was originally published on ScrumAlliance.org]

Transforming Agile Nay-Sayers Into Enthusiasts

written by Manoj Khanna | January 2, 2017



Agile is increasingly mentioned as the go-to method for product development, and given the coverage on agile, it appears that there is a consensus that agile is at least viewed in a neutral light, if not favorably. Despite this, there *are* adamant nay-sayers against agile. For those who are attempting to transition their teams into embracing agile, it can be difficult if a team member is resistant towards agile. This post serves to provide insight into the criticism that some may have towards agile, in order to assist those seeking to convert agile nay-sayers into enthusiasts. For those who hold an unfavorable view of agile, this post will detail the potential of using agile with customer insights to transform

products, along with the top agile practices to adopt in order to maximize a product's reception.**Main Criticism Against Agile***Lack of Structure*
A common criticism against agile is the lack of structure, especially in comparison to traditional methods such as waterfall. Indeed, agile is more open-ended and it embraces changes. That is not to be mistaken with chaos, though. Critics may misinterpret the lack of structure to lead to team members working on any number of tasks that may be irrelevant, and that progress cannot be achieved efficiently. Agile, in its lack of linearity and openness to quick changes, induces the opposite effect. It enables more progress to be attained during development, as multiple rounds of testing enable product features' issues to emerge quickly and to be addressed immediately, resulting in a more complete product.

Rushes Into Development

Some may view the multiple cycles involved in agile to be a "rush" and that it undermines thorough and successful product development. Agile cycles consist of the stages of more traditional methods, but less time is spent on each stage within each sprint. This "rush" into the next stage is precisely what enables agile product development to incorporate so much user feedback into the process – and this incorporation of feedback results in a better product.

"Agile Fever"

Another main criticism against agile is the apparent "agile fever" that is sweeping across businesses and industries in the attempt to benefit from this method. This criticism is not unwarranted, for as is true with anything, too much of a good thing can be detrimental. With agile, it is important not to rush into implementing it merely because everyone else appears to be doing so; it is vital to thoroughly understand agile before adopting it, and even upon adoption it, the process should be tailored to each business individually.

Using Agile With Customer Insights to Transform Products

Agile, contrary to the main criticisms that exist, is an efficient method to transform products into ones that are well-received by the targeted customers. Each sprint in agile product development provides the opportunity to glean and incorporate user feedback into the next sprint. Not only is user feedback allowed to be a major factor in the product's development, but agile also provides ample opportunity for product issues to emerge and to be addressed on the spot, before the final product is released. Under agile, the product is completed multiple times and assessed as such, allowing for it to be enhanced a number of times more than if another method were used.

Each sprint in agile product development can be viewed as a trial run, wherein user assessment is gleaned and addressed accordingly. Had the product been developed under a method other than agile, user assessment would not be obtained until the final product was released – by which time it would be more costly to fix the issues and to incorporate what users want and need from the product; the product's reception and success would suffer accordingly.

Top Agile Practices to Adopt

To maximize the potential held by agile product development and customer insights, it is important to emphasize the adoption of certain agile practices, namely continuous integration and design review. Continuous integration of feedback results from, and fuels, constant effort to glean feedback on and enhance the developing product. This is vital in creating a more successful final product that matches or surpasses user expectations. Design review, the other top agile practice to adopt in order to maximize integration of customer insights into the final product, enables teams to review design stories with consideration of the latest product feedback; it poses the opportunity to plan further work on the

product with the feedback in mind.

There will continue to exist agile nay-sayers who will not embrace agile, despite the lack of evidence for some of the main criticisms against agile. For those who are swayed by the potential held by agile to incorporate user feedback into the creation of successful products, there exists ample information for them to begin their agile journey.

{image courtesy: flickr/JD Hancock }

How Measuring Velocity Helps in Your Agile Journey

written by Manoj Khanna | January 2, 2017



Measuring velocity is a useful way of determining how long an agile project will take to complete, by providing a rough estimate of the amount of work the team can complete in a sprint. It is necessary to keep in mind, however, that as insightful as knowing your team's velocity is, velocity is *not* a true measure of your product's development.

This post will discuss what velocity is and the reason why we measure it, and why management is interested in higher versus lower velocity.

Velocity Defined

Velocity, in terms of agile product management, is a metric that provides insight into approximately how much work a given team can complete in a sprint. Measuring velocity uses information from a completed sprint, so if your team is approaching its first sprint, you must know how many people will be involved in the project, the maximum amount of work each person can complete, and the total number of workdays in the sprint, in order to calculate the velocity.

Calculating velocity involves units of work that can be defined as hours or story points – a metric capturing the complexity of implementing a story – for example, and tasks. Velocity is calculated by adding up the difficulty metric of every backlog task, such as stories, completed by the team in a given sprint with the units of work for those tasks. This provides the velocity in terms of units of work per sprint.

Why Measure Velocity?

Velocity provides an estimate of how much work your team, specifically, can complete in a given sprint. As velocity is calculated using the work of previous sprints and if everything remains constant – such as the same people are involved and the tasks are relatively of the same nature – then velocity is consistent. If the team had a velocity of 30 story points per sprint for previous software development projects, then it can be expected that, all else constant, the team will have a velocity of 30 story points per sprint in the next software development project.

Velocity is useful in estimating the approximate length that a project will take, as well. If the project at hand has 120 story points' worth of stories, and previous sprints show that the team has a velocity of 30 story points per sprint, then it can be expected that the team will complete the present project in four sprints.

Higher Velocity Versus Lower Velocity

As velocity signifies an agile team's productivity – the velocity value indicates that either the team completed fewer high-difficulty stories, or they completed many stories of lower difficulty – it is more desirable for a team to have a higher velocity than a lower one. A higher velocity would indicate that the team is capable of completing more stories or high-difficulty stories, which translate into more progress towards the project.

It is worth noting, though, that when teams are measuring their velocity at the start of a project, the velocity value will not be as accurate as it will be for later sprints. An underestimation when initially measuring velocity can result in a higher velocity later on in the project. Similarly, an overestimation in initially measuring the velocity will lead to a lower velocity later on in the project, as the velocity is adjusted with each completed sprint. As such, the velocity value that is calculated later on in a project may not accurately reflect the team's productivity; their productivity could have remained constant, yet the change in velocity value may be due to an under or overestimation.

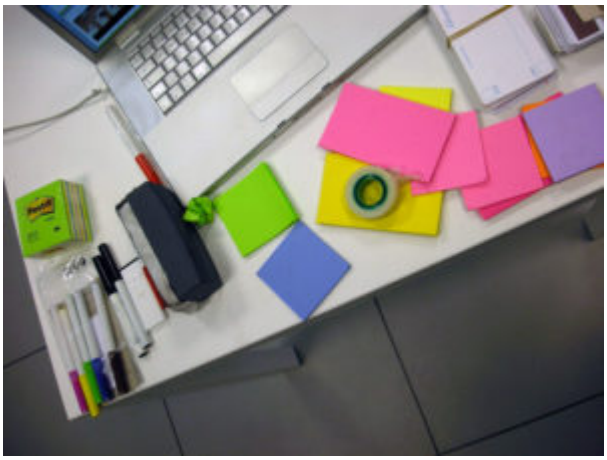
Velocity Is Not a True Measure of Product Development

As useful as measuring velocity is, an agile team should not rely on their velocity to indicate the progress of their product's development. The velocity is measured based on each sprint, and with each new sprint, new product features' information is accumulated and added; this changes the stories and story points associated with the next sprint, so the previous sprint's velocity is not necessarily indicative of the team's work in the next sprint. Many new features may have to be added or changed in the next sprint, so the nearly completed product of the previous sprint may actually be 40% completed, given the previous sprint's feedback. Even though the team had a high velocity in the previous sprint, product feedback necessitated even more work before the product can be deemed complete.

{Photo courtesy: Flickr/Jan-Hendrik Palic}

Reasons Why Agile Coaches Must Get Their Hands Dirty

written by Manoj Khanna | January 2, 2017



Agile coaches must be involved in the client's Agile process. Those who have the impression that coaching can be done from the sidelines are mistaken, for coaches must get their hands dirty if they want to bring about successful Agile adoption.

To better explain why Agile coaches cannot be observers, I will provide details about the possible mayhem within the Agile landscape, some twisted thoughts about coaching, and the high-level transition plans that Agile coaches should set in motion for their clients.

Mayhem within the Agile landscape

The landscape is rampant with companies trying to transition their teams to Agile while not implementing the method fully. Companies transitioning from a traditional method to Agile face obstacles in the process. Company leaders must fully understand Agile. On top of that, they then must convey the

information to their teams and get everyone to coordinate. During the Agile transition, they may encounter issues such as working on too many projects at once, improperly allocating resources, and not forming truly cohesive teams. In certain companies, such transitions are major ones – a situation that increases the difficulty of transitioning to Agile.

These obstacles are such that the move toward Agile likely will not be smooth and successful the first time around. Without Agile coaches to guide them, companies will fumble through trial-and-error projects numerous times before they attain full Agile implementation. Their Agile adoption process will require more time than if they had the assistance of an Agile coach.

Twisted thoughts about Agile coaching

Not any Agile coach will do, however. Companies who decide to enlist the assistance of a coach need help in laying a solid Agile foundation and smoothing their adoption process. In order to provide such assistance, Agile coaches must immerse themselves in the process as if they were part of the company.

Too many Agile coaches, however, are under the impression that coaching consists merely of teaching the company leaders the Agile components. They teach Agile instead of walking the companies through the process as coaches. Few deem it necessary to observe the leaders and teams or to provide insight into what is properly implemented and what needs to be done better. Yet Agile coaches cannot be sideline observers of their clients, because Agile implementation does not come with a guidebook. Each company will have a unique implementation process based on individual dynamics, and coaches need to understand each company and tailor their coaching appropriately.

The Agile coach’s necessary involvement

Agile coaches must coach hands-on. Companies know their transition to Agile would take significant time for them to sort through on their own, and often they want to thoroughly adopt Agile more quickly. Such companies enlist Agile coaches for their seasoned insight and experience, which coaches can provide if they work properly within the company.

Coaches who stand on the sidelines, give advice on the Agile method, and then leave quickly do not assist much toward a successful transition to Agile. They need to remember the reason why the companies sought them out in the first place – to guide them through the Agile implementation and provide pointers along the way. Agile coaches need to understand the companies that they work with, provide insight into how Agile can fit into each company individually, and guide each company through the transition.

Agile coaches can become involved at any stage of the transition process. It depends when a client has deemed it necessary to enlist a coach’s assistance. Ideally, a company would enlist an Agile coach from the beginning to reduce delay.

Transition plan

Agile transition is not a one-size-fits-all proposition; there is not one structured set of rules for the process. To assist companies in attaining the expected outcome of a quick, smooth Agile adoption, coaches should have transition plans tailored to each company. However, coaches can approach this in a general order, by first identifying and addessing the company’s particular issues. For instance, if a company’s team members do not work cohesively, then the coach should guide the leaders to focus on removing the impediments to good teamwork. Coaches must help pinpoint and address issues impeding Agile implementation.

This entails providing on-the-ground assistance in coordinating teams throughout the company to understand and use Agile, defining each company's process, aligning the entire organization with Agile, and guiding the company to see issues and solutions more quickly.

Summary

The key is that Agile coaches need to be part of their clients' Agile process. When coaches see themselves giving advice without taking time to observe the client's teams on multiple occasions, that is when they cease to be Agile coaches.

{Note: This article was originally published on Scrum Alliance. Image courtesy: Flickr/Bikolabs}

How to select an Agile tool that works for you and your team

written by Manoj Khanna | January 2, 2017



The agile approach to project management entails constant

changes which require a set of tools that are not only resilient to the various changes that will emerge in the development process, but which also suit how your team works. The ideal tool for each organization will be different, but in any case, it will facilitate agile work processes and allow for team structures to be adapted for maximum project results.

Read more here.

{image courtesy: Flickr/Dushan}