Velocity
PARTNERS

A Scrum Team Booster Shot

Table of Contents

All the articles in this document have been originally published in Velocity Partners Blog

Velocity Partners is committed to continuously improving our agile practices and those of our clients. We regularly add posts to our blog (at http://www.velocitypartners.net/blog) to contribute back to the community at large. We hope that by sharing our experiences and expertise we can help individuals and teams on their own agile journey.

In this paper, we highlight some blog posts from our former Principal Agile Evangelist Bob Galen. Bob is a recognized leader in agile practices and these posts can act as a "booster shot" for any Scrum Team. The guidance Bob provides is useful for anything from a new team just learning the ropes to an experienced team who may be starting to plateau.

Thank you for downloading this "booster shot" and I hope it helps you and your teams succeed.

Bill DeVoe

Principal Agile Evangelist

Velocity Partners

The "3 Amigos" in Agile Teams

Article written by Bob Galen, an Agile Methodologist, Practitioner & Coach based in Cary, NC. In this role he helps guide companies and teams in their pragmatic adoption and organizational shift towards Scrum and other agile methodologies and practices. Contact: bob@rgalen.com

I think it was George Dinwiddie that first coined the term "3 Amigos" in agile development around 2009. The analogy was akin to the movie from the mid 90's by the same name. The Amigos in the agile sense were functional roles:

1. Developer(s);
2. Tester(s);
3. and the Business Analyst or the Product Owner.

It could literally mean more than three as well. The point was, balanced collaboration in agile teams across those three roles. George was alluding to these roles from an Acceptance Test Driven Development (ATDD) perspective. He wanted these three constituencies to be heavily collaborative (have conversations) around the Acceptance Tests or Acceptance Criteria for each user story.

In my classes, I literally explain this as an "extension" of agile team pairing—in that I'd prefer the pairs to be converted to "Triads" around each story. Believe it or not, Ken Pugh leveraged the term Triad in his book on the same "ATDD-ish" driven topic (see the references).

3 Amigos Meeting

Recently I've heard of teams scheduling 3 Amigo style meetings as a part of grooming their backlogs. The idea is to create a checkbox for each user story where it has to be "3 amigo-ized" before it's deemed ready for sprint execution.

I explored this notion of readiness in a previous blog post.

While I like this idea, and I certainly think making it a part of "Ready" is quite creative, I'm slightly troubled by the practice. I personally think the 3 Amigo's analogy is less of a meeting and more of a collaborative mind-set.

So when do the "Amigos" meet?

Certainly they meet as part of Product Backlog development and grooming. I think that's a given. But, I'd like to say continuously as well, for example:

- Collaborating in the Sprint Planning meeting around how the stories fit into the Sprint and what the Sprint Review will look like from a cohesion perspective.
- Chatting when each story is "first picked up" within the Sprint to ensure the team understands the nuance of the story and how to effectively design and test it.
- During story development, the Amigos periodically check in together to ensure the story is on-track. Usually demo's of partially completed code drive the discussions;
- Once the story is complete, the Amigo's gather around it and execute it. Going over acceptance criteria and other attributes, until the Product Owner is satisfied and "signs off on" the story;
- But their role isn't quite over. The Amigo's play a strong part in demonstrating the story in the Review, gathering feedback, and digesting whether the story is truly complete. If so…they move onto the next, and the next, and…

What it's not…

At the risk of "picking on" Jon Kruger, I want to use part of his description to make a point:

Outcomes

The primary outcome of the three amigos are acceptance tests written in Given/When/Then format. Actually writing these out can take a little time, so we don't do that with everyone sitting in a room. Typically a developer or a tester will work on it outside of the meeting, and once they have the scenarios all written out, then we do a quick review with everyone else that was involved in the original three amigos meeting to make sure that we all agree with what was produced.

I beg to differ slightly with Jon. His description is much more focused on the 3 Amigo's as:

- implying a meeting with specific, process-driven activity;
- implying exhaustive scenarios, written in advance;
- implying a prescriptive format of Given-When-Then;
- implying sign-off on acceptance tests, prior to the story entering the sprint.

While the meeting itself is good, what he and many others seem to miss is the mind-set part of the Amigo's that is continuously happening around the team. Please don't loose that focus in your thinking. And it certainly doesn't have to use Given-When-Then format for every "conversation"

Wrapping up

George, Ken and many others have struck on the collaborative power of these three roles within agile teams and their work. Don't get caught up on it implying a meeting or any other ceremony, although those things may be outcomes of it.

Instead, consider it a role-based reminder of who needs to collaborate around the work that agile teams are producing continuously. Also think of it as a pairing surrounding sprint activity (the work) rather than a meeting-based activity (talking about the work). I think that was always the primary intent.

But please, do talk about "Amigos collaboration" early and often within your teams. The real point is that you'll build better, more valuable, and more relevant software as a result. Now isn't that something worth hollering about?

Stay agile my friends,

Bob.

Note: the picture can be found here: http://en.wikipedia.org/wiki/File:Healthywealthy.jpg

References

- Ken Pugh's book – http://www.amazon.com/Lean-Agile-Acceptance-Test-Driven-Development-Collaboration/dp/0321714083/
- 2 references to the notion of a 3 Amigo's Meeting: http://jonkruger.com/blog/2012/01/04/the-three-amigos/, https://www.scrumalliance.org/community/articles/2013/2013-april/introducing-the-three-amigos
- Video – George Dinwiddie: http://www.infoq.com/interviews/george-dinwiddie-three-amigos
- Podcast – George Dinwiddie: http://agiletoolkit.libsyn.com/tips-and-advice-acceptance-test-driven-development-and-the-3-amigos-process
- Article on Better Software in 2011, by George Dinwiddie: http://www.stickyminds.com/better-software-magazine/three-amigos

Agile Testing — The Agile Test Automation Pyramid

Article written by Bob Galen, an Agile Methodologist, Practitioner & Coach based in Cary, NC. In this role he helps guide companies and teams in their pragmatic adoption and organizational shift towards Scrum and other agile methodologies and practices. Contact: bob@rgalen.com

In this post, or perhaps truly an article, I want to explore a common approach for implementing an effective strategy for your overall agile automation development. It was Mike Cohn who established the initial model of a pyramid view towards traditional and then agile test automation.

This model is fairly widely known as the *Agile Test Automation Pyramid**[1]*.

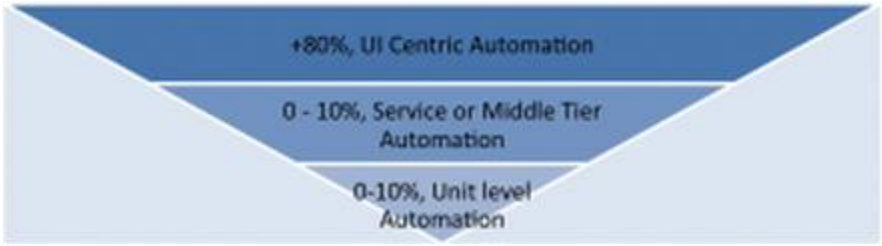

Figure 1, Traditional Test Automation Pyramid

In the traditional view, most if not all of the effort was in developing UI-centric functional tests that explored the application via the GUI. There might be some lower-level tests and a few unit tests, but teams mostly staid at the upper tier.

Why?

For several reasons, but first and foremost, the testers were the ones primarily writing the automation, so their comfort zone was towards functional testing. It didn't help that the majority of the automated testing tools were focused towards leveraging the functional UI as the point of entry.

Developers operated in the service and unit tiers, but typically didn't invest much of their time in developing automated tests. There was little 'value' in it from their perspective, when they were being driven to deliver the features by prescribed dates.

But this strategy is flawed. It's inherently unstable and brittle; as the application (UI) changes the automation is nearly always impacted. Therefore one problem is automation stability and ongoing maintenance costs. Another problem is that it doesn't engage the entire team. Instead, only the testers typically develop it and they usually only comprise a minor portion of the team. Quite often they also become overloaded with time spent on testing versus automation development.

The agile test automation pyramid is a strategy that attempts to turn all of this around–literally.

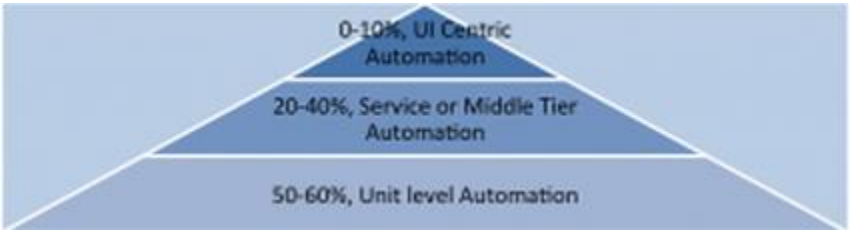

Figure 2, Agile Test Automation Pyramid

Turning Things Around

The first change is taking a whole-team view. Instead of the testers being responsible for testing AND writing all of the test automation, it becomes a whole-team responsibility. The developers take most of the ownership for unit-level automation, but testers can operate here as well. The upper tier focuses on limited UI-based automation. Usually, these are longer running, core customer usage workflows that are best implemented at this level.

The testers typically operate on these, but there are very few tests within this tier. And remember that the developers can operate here as well. The two layers are met by middle-tier automation. This is often the domain of the open source ATDD/BDD tools, such as: FitNesse, Cucumber, JBehave, Robot Framework, and others.

One key thing to note is that traditional automation was often a one-tool operation, with Mercury/HP commercial tooling (Quick Test Professional or QTP) leading that space. The agile approach is tool agnostic, but also aggregates tools that are appropriate for each layer. Therefor no "one size fits all" thinking is allowed. For example, as of this writing, these are common used tools at each tier:

1. **UI tier:** Selenium, Watir, or traditional QTP
2. **Middle tier:** FitNesse, Robot Framework, and Cucumber
3. **Unit tier:** xUnit family variants for example JUnit or NUnit for Java and .Net respectively

The other consideration is that there are extensions to many of these. For example, both Robot Framework and Cucumber have Selenium plug-ins so that they can 'drive' the UI as well as the middle tier. This implies that the middle tier tooling, and automated tests for that matter, can extend or blend into the lower and upper tiers.

Advantages?

The advantages of this approach mirror the disadvantages of the traditional approaches. First and foremost, if you take a whole-team approach, you now have automation work crossing every team member—so your capacity is increased. You also get better automation designs and improved testability because developers are weighing in.

Another important benefit is maintenance. Often it becomes a team-wide responsibility to keep up with automation maintenance, both the infrastructure and the tests, as the team is developing functional user stories. I like like to make it part of the organizational and teams' Done-Ness Criteria, for consistency and rigor. For example, a story is not "done" unless all of the automation associated with it, new and old, is working correctly (passes).

An advantage that I didn't emphasize in the traditional case is automation coverage. One could argue that automation from the GUI down was a limiting exercise. There was always back-end functionality and business logic that was incredibly hard to "reach" via the automation. And internal infrastructure was nearly impossible to exercise / test via automation. In a multi-tiered approach, you should be able to "reach" any behavior or functionality that the team deems to have value to automate.

Two Final Cautions

I've found it surprising but true that many developers don't really understand (how) to write effective unit tests—especially when you expect those tests to be part of a cohesive whole. Partnering with testers can help immensely when designing unit tests, even if the testers can't program in the language du jour. I've found that training can really be a force multiplier here. That would include a 3-4-5 day class focused on OO design skills, patterns, refactoring, and TDD. That combination can be incredibly helpful in creating a baseline of skill across your teams.

And one more final caution—don't automate everything! The strategy or business case for 100% automation is nonsense. In fact, I'm notoriously against using any sort of "magic number" when setting goals for any sort of automation level or coverage. Rather, ask the team to look at each user story and determine the "right

level" of unit, middle-tier, and UI tests in order to adequately cover it compared to its complexity and value. THEN, implement those tests as part of your Done-Ness criteria. So every story should have a different level—determined by the team. Point being—trust your teams to determine the right balance between automation and other relevant forms of testing.

From an Agile Testing Perspective

I like to think of the agile automation triangle as THE model for automated testing strategy within the agile organization. While the execution of the triangle is a whole-team activity, the QA and Test team leaders and the testing community should spearhead the automation strategies themselves.

This involves test Architects leading the efforts to evangelize the strategy and approach and to guide their organizations towards effective tools selection. Often, having a tooling "bake-off" is an effective way to garner whole team awareness, interest, and feedback in the selection process.

A Real World Example

In Figure 3, you see an example of our implementation while I was at iContact. I literally 'plucked' this snapshot from a real-time status wiki page. It was where we stood in actual test counts in mid-2012.

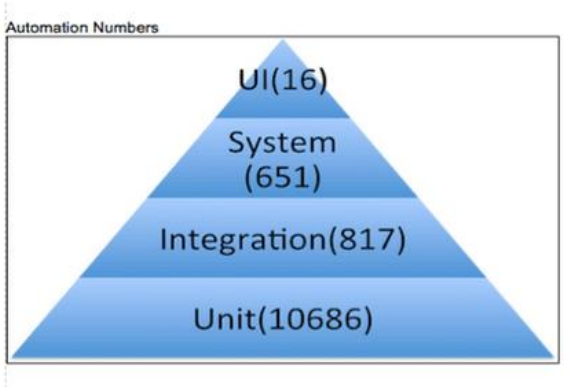

Figure 3, iContact Automation Pyrimid circa (2012)

In our case, we leveraged the 3-tier model, but we differentiated between System-level and Integration-level tests in the middle tier.

Our view was that Integration-level tests were tests implemented in Cucumber, our tool choice in the middle tier, but they were focused more towards unit-level testing. Another way of saying that is that they were larger scale unit tests. Our System-layer tests were true middle tier tests that were focused more on functional behavior. Of course, there was a "fuzzy line" between the two.

I've found this is often the case in real world projects, that the pyramid boundaries are not always clear-cut. In our case, we defined categories that worked for us and allowed us to report out on the execution of the various layers.

Another important point, everything below our UI level tests were intended to be run on a "check-in basis", that is, when every developer performed a check-in, approximately 12k tests would run to provide them feedback as part of our Continuous Integration tooling and virtualization environments. So the tests needed to be developed with performance in mind.

We were constantly tweaking our tests to bring the execution time within reasonable constraints for our team. Usually are targets were between 15-30 seconds of run-time. This was based on our commitment to run as many tests as possible on finely grained code chunks so that we could get real-time feedback.

It was hard work to keep it tight, but well worth the efforts.

Wrapping Up

I've been recommending and personally leveraging the concepts behind the Agile Testing Pyramid since roughly 2005. It's driven my testing strategies at two companies–ChannelAdvisor and iContact. I've also made it a strong part of my agile coaching when I'm influencing my clients' testing strategies. I'm still shocked that many in the testing community are still taking a monolithic, UI centric view towards developing their automation. I just don't understand it.

What are some of the advantages to the approach?

1. Whole-team automation
2. Distributed maintenance
3. Lower cost
4. Run-time speed (allowing for more coverage on a finely grained check-in level)
5. Consistent coverage and ongoing feedback

At Velocity Partners, we've implemented a wide variety of test automation strategies & frameworks with our clients. We've used many of the open source tools. As we've worked, we've discovered what works and what doesn't. Almost to the point of becoming a SME when it comes to agile test automation. If you're struggling with implementing agile test automation effectively, and its a hard problem to solve, you might want to consider a "partner"

Stay agile my friends,

Bob.

[1] There is not a good reference for the original proposed idea. In my research, it appears to pre-date 2008 and a conference presentation surrounding it. You can see Mike referencing the idea here in another context.

Characteristics of High-Performing Agile Teams: What are they?

Article written by Bob Galen, an Agile Methodologist, Practitioner & Coach based in Cary, NC. In this role he helps guide companies and teams in their pragmatic adoption and organizational shift towards Scrum and other agile methodologies and practices. Contact: bob@rgalen.com

Jeff Sutherland used to characterize gaining Scrum or agile maturity as a team reaching a high-performing (or hyperproductive) state. He shared stories around these teams on his blog and via conference presentations (*see references*). He's making the point less today than he was a few years ago, but agile maturity and the inherent results, is still an interesting question.

One of Jeff's colleagues in 2008-2010 timeframe was Scott Downey. Scott is an experienced agile coach. Here's the intro to a blog post and paper that he developed in 2008:

Scott Downey, MySpace Agile Coach, has a way of bootstrapping Scrum teams to a high performing state in a company that is about 1/3 waterfall, 1/3 ScrumButt with project managers, and 1/3 pure Scrum with only Scrum roles. Scott consistently takes teams to 240% of the velocity of MySpace waterfall teams in an average of 2.9 days per team member where the team includes the ScrumMaster and Product Owner.

So I thought in this blog post it would be interesting to explore the notion of a high or hyper productive agile team. What does it look like? Is it the goal? Does pure productivity matter, and if so, what does delivery look like? Is it sustainable?

I'll start with sharing three of my own indicators and then punt the ball to all of you.

3 of my own - Swarming

It's so incredibly easy for agile teams to work on too many things at once. To focus on keeping individuals busy, for example making sure the developers are coding, rather than focusing on getting Features-Stories-Work…done. So instead of serially working and handing things off to different functions, you'll get phenomenal throughput by swarming around the work and focusing on getting it "Done" as soon as you

possibly can. The Kanban notion of a WIP limit (work in progress) reinforces this notion of multiple team members (with different skill-sets) combine on the work.

Stretching, trying new things, and occasionally…failing

Too many agile teams "play it safe". I see teams all of the time who worry about their Sprint commitments in Sprint Planning sessions. They calculate hours and seconds and only commit when they feel it's a safe bet that they can deliver the work. I'd much rather have then trust their team and their gut feel…and give it a go. It makes me smile when teams plan "stretch items" as part of their Sprints. It tells me they're attempting to go "above and beyond" what they perceive as their capabilities and capacity. I like that. Sure, sometimes they fail to deliver. So, what? Often they exceed everyone's expectations.

Courage

Courage is one of the five core Scrum values or attributes. To me indications of it include: less filtering of communication, telling it like it is, and challenging each other as team members. A big part of this is humility—to show a willingness to say "I don't know" or show a sense of vulnerability. Often it includes the ability to admit it when you need help and not waiting too long to ask for it. And courage extends beyond the team to organizational leadership who show the courage to trust and support their teams. It's easy to "go along with the flow". It's hard to truly be part of a high-performance, self-directed team and to engage in your teams overall dynamics and continuous improvement efforts.

Now it's your turn

I want to encourage comments where you share your own ideas as to what are the key ingredients to making a high/hyper performing agile team. Please consider a broad view that might encompass:

- Technical skills & practices
- Soft skills
- Behaviors
- Specific agile techniques or practices
- Tooling

And if you don't have a pattern, but know what inhibits high-performance (an anti-pattern) then please share those as well. In fact, I often learn more from "what not to do" when I'm coaching. Thanks for sharing with the community. Stay agile my friends,

Bob.

References

- http://scrum.jeffsutherland.com/2012/12/update-excel-spreadsheet-for.html
- http://scrum.jeffsutherland.com/2010/10/scrum-metrics-for-hyperproductive-teams.html

What does an Agile Coach do?

Article written by Bob Galen, an Agile Methodologist, Practitioner & Coach based in Cary, NC. In this role he helps guide companies and teams in their pragmatic adoption and organizational shift towards Scrum and other agile methodologies and practices. Contact: bob@rgalen.com

If you're not aware, one of my certifications is as an Agile Coach, or more specifically, a Certified Scrum Coach (CSC) by the Scrum Alliance. At the time I received my certification in 2012, I was the 47th person who had earned a CSC in the entire world. Now I think there may be approximately 60 of us. So it's an exclusive certification and fairly hard to achieve.

The Scrum Alliance positions the CSC on "equal footing" with the CST or Certified Scrum Trainer certification. The latter is focused towards training new Scrum Masters (CSM) and Product Owners (CSPO). While the CSC is focused towards the "doing" of Scrum in real-world organizations. Often the term Enterprise or Organization is aligned with the CSC coach due to the fact that they're coaching organizational transformation towards agile approaches as much or more than the individual teams.

But Bob, what do they do?

In fact, this is the focus of the CSC – that of organizational coach. So getting back to the title, what does an agile coach do?

The first thing they do is teach. Now it may sound like this conflicts with the CST, but in reality they compliment one another. Every good coach I know has a "toolbox" of classes that they can deliver for their clients. Those classes might go from basic introductory material to specialized classes for agile specifics. I for example, have a unique interest in Scrum Product Ownership, so I've developed material and approaches for teaching:

Another area of focus is simple observation within context. That is, the coach visits or lives with the organization and teams for awhile. They attend meetings and engage with the teams. They observe and gather insights into areas where the teams are strong and where they might have adjustments to make. Some coaches couch this with a more formal assessment model. Others just observe and make recommendation on-the-fly. Either one works fairly well.

Why do we need a Coach?

Well the short answer is that…you don't. Many more agile teams do without coaches and they do reasonably fine. I liken it to acceleration. A seasoned agile coach can help guide a team to a much faster ramp-up in their

agile adoption and overall performance. Instead of "going it alone" and "figuring it out along the way", a coach or guide can truly make a difference.

The other critical drive is the lack of concrete documentation or guidance in the agile methods themselves. It's not like you were taking on Waterfall variants or Rational Unified Process. These have hundreds or thousands of pages of templates, checklists, and documentation that tells you what to do. Agile isn't like that. It's emergent, team-based, and situational or context-driven. There is no defined "Play Book" for all organizations. That implies that you have to largely figure things out as a team.

While that's a great approach, having a coach can make that learning curve much faster.

Isn't the Scrum Master a Coach?

Well, yes and no. I would agree that the Scrum Master is the coach of their respective teams. That would be a part of their role. But it's team centric or focused, and another part of a coaching role is to move the entire organization. I would include your leadership team as part of that movement and cross-functional team implications. An agile coach should be thinking, operating, and influencing much more broadly than the individual Scrum (or agile) team.

In fact, I think the better coaches take on a "Coach the Coaches" mindset and model, so they would be coaching the Scrum Master, Product Owners, Functional Managers, and Organizational Senior Leadership. So the focus is more upward and laterally (80%) than downward (20%) toward the teams themselves.

Wrapping Up

If you're in the market for a coach, I've written a guide to selection criteria that I've used when finding and hiring agile coaches. I've tried to make it as "balanced and even handed" as I could. Not only can it help you interview and find a coach, but it should provide more guidance on what the critical skill areas are. You can find it here.

So ultimately, what does an Agile Coach do? They help you deliver the good for your clients. It's as simple as that.

Stay agile my friends,

Bob.

Where to go next?

You can visit our website at **Velocity Partners** and read more articles in our blog here.

Velocity Partners is a nearshore software development company that strategically aligns with companies to deliver results better, faster and more cost effectively. It does this by leveraging a distributed Agile model. Based in Seattle, **Velocity Partners** operates software delivery centers in Argentina, Uruguay, Colombia and Venezuela where it has found the culture, talent and business mindset to exceed its standards for success.