Better Together

Linda Wilkinson

linda.wilkinson@evisions.com

Abstract

So you're working on or with agile teams – but are you really agile? Or are your sprints more like little waterfalls, with testing **still** at the end of the process, **still** dependent on one or two people doing the testing, and **still** generating too many bugs too late to fix on time – forcing painful choices between quality and schedule? Too many bugs found in Production in spite of your spiffy TDD processes? Are you automating a test set that wasn't that great to begin with? Have a test team using exploratory testing when they don't know how to explore?

Part of our mandate as testing experts is to raise the quality bar. How do we help make that happen? How do we improve and educate ourselves and our companies in regards to testing?

One of our big challenges as testers is recognizing that maybe, just maybe, we aren't "all that". Our testing is by necessity limited in scope, perspective, and experience. We often do not play roles as the developer, end user, or devops. If testing is solely up to us, we will miss things. Important things. The purpose of this paper is to talk about ways we can collaborate to build (and therefore automate) better test sets. How to negotiate who does what and make testing a team activity. How to move towards or improve TDD (Test Driven Development) efforts. And above all, how to do that quickly, collaboratively, and with an eye towards improving quality.

Biography

Linda Wilkinson is Director of Quality Engineering for Evisions, Inc. She has been in the QA/QC field for over 30 years. She holds many certifications (some on tools that no longer exist!), has been a speaker and author, is PMP trained, and has acted on expert panels for process improvement and quality metrics. She is passionate about the quality field, and is dedicated to solving complex problems with simple solutions, mentoring, and learning/brainstorming about new ways to work.

1 Introduction

Agile methodology is no longer new. Many of the test professionals working in the field right now have never used anything **but** Agile technique. As with all project methodologies, however, the definition and methods of implementation for *agile* varies. So what, exactly, is "Agile"?

Agile project methodologies are specifically designed to allow project teams the latitude, independence, empowerment, and authority to create products quickly and responsively through a variety of collaborative techniques. It doesn't matter if it's Kanban, Scrum, Lean, TDD (Test-Driven Development), Iterative, or something homegrown that falls in-between, agile methodology involves working together to get something done efficiently and effectively.

What sets successful projects apart from unsuccessful projects is the team, not the methodology. Specifically, highly collaborative teams are more successful than those who are not collaborative. Many would argue that agile methods require collaboration, and it's true that agile methodology encourages collaboration and therefore tends to push project efforts toward success. At the same time, many agile projects miss their deadlines, do not produce what they were hoping to produce for their sprints, and produce code with varying levels of quality. Some agile teams are superb, some are mediocre. Yet they can be following exactly the same methodology. How can that be? Well, collaboration is, to a large degree, an intellectual concept. While it can be encouraged, it is difficult to demand or measure. Collaboration is not a process; it's a mind-set. A positive, shared mind-set working towards a mutual goal.

While project methodology has been encouraging and educating agile teams to become increasingly collaborative, the testing processes for agile projects have not evolved at the same rate as other agile practices. The purpose of this paper is to present the concept of collaborative testing. The techniques themselves are methodology-agnostic and could be used successfully on any type of project, but this paper will focus on incorporation into agile methodology, as the level of collaboration required is complimentary in concept and thus more easily absorbed into agile teams.

2 What problems are we trying to solve?

In a truly agile team, the lines between roles becomes blurred. Members of the team can and do act in any capacity in order to get a given task completed. Oddly, that often doesn't apply to the testing tasks. In many organizations, the development staff turn their completed code over to a testing resource, who develops tests, performs the testing, notes defects, and returns results back to development staff. What's wrong with that picture? Well, it's not really *agile*, is it? It's actually a mini-waterfall, with testing taking place at the end of a sprint (or with its own dedicated sprint), reporting errors at the end of the process, and often delaying either the sprint underway or impacting the start of the next sprint.

In some teams, the QA resource can handle the test automation for the team, automating the gamut of tests from unit through acceptance testing. In this case, the QA resource is truly an *SDET* (Software Developer in Test), with all of the issues in regards to time and perspective experienced by any other developer on the team. Every minute spent automating a test is a minute that is not spent thinking about, exploring, or designing a test. In other words, where this process often fails is in the analysis of what is needed to adequately perform the testing, not the actual automation or testing of the change. The test set itself is incomplete. This can lead to "passed" code that doesn't perform well in the field.

One individual, no matter how talented, and whether a developer, QA resource, or BA (Business Analyst)/PO (Product Owner), has one perspective. In addition, we all have our individual biases. Teams often have issues where (some) testing performed downstream resulted in unexpected error because the code had not previously been used, thought of, or tested quite that way. Again, the test set the team was working with was incomplete.

Overall, the problems we are trying to solve are those that have challenged us for many years. Too many problems found too late in the process.

3 Roadblocks

People often don't like change and generally they don't appreciate anything they particularly value challenged. These ideas will challenge what is considered *typical* about some agile team practices, particularly in the area of acceptance criteria. Part of my mandate is to support **any** project with testing services that are both efficient and effective. For that reason, I am completely indifferent to the methodology chosen. Others are incredibly passionate about their methodology, are extremely attached to their ceremonies and current methods for doing work, and it will require some significant negotiation on your part to get them on board with trying something new. What I like best about agile methodology, personally, is that in its purest form, it encourages experimentation and creative thinking. Few companies actually practice that, however. It can be very difficult to get people to think about something they've done one way (perhaps for their entire careers) in new and different ways. They **love** their established way of working. It's comfortable, like an old shoe. What we are trying to do here is to encourage them to throw their old shoes away. The ease at which this is possible will depend on how comfortable that shoe really is and whether it is actually falling apart.

It helps that collaborative testing is complimentary to other accepted agile practices and encourages collaborative behaviors. At the same time, it is a new way to work and it has yet to go through the growth and change of other agile processes. Collaborative testing is an advanced topic for several reasons. First, it requires someone in a position to act as a change agent to get everyone on board and make it happen. Second, the collaborative, intellectual nature of the technique requires a level of sophistication that some organizations may simply not possess. Yet. If your company works with a variety of structured brainstorming techniques and practices, this process will come very easily to you. If not, the process can be like pulling teeth at first because you'll be teaching people to be collaborative in a group. If your company is not accustomed to structured brainstorming technique, it might be helpful to start with something simple, like introducing the idea of "lean coffee" (Modus Cooperandi 2015).

4 Let's rethink "acceptance criteria"

.What is acceptance criteria? This definition was pulled from scrumalliance.org:

***Microsoft Press** defines Acceptance Criteria as "Conditions that a software product must satisfy to be accepted by a user, customer or other stakeholder." Google defines them as "Pre-established standards or requirements a product or project must meet.".*

Acceptance Criteria are a set of statements, each with a clear pass/fail result, that specify both functional (e.g., minimal marketable functionality) and non-functional (e.g., minimal quality) requirements applicable at the current stage of project integration. These requirements represent "conditions of satisfaction." There is no partial acceptance: either a criterion is met or it is not.

These criteria define the boundaries and parameters of a User Story or feature and determine when a story is completed and working as expected. They add certainty to what the team is building.".

It's generally agreed that acceptance criteria are not as detailed as tests. The criteria are often written by the author of the user story. Often the author wants to keep the criteria to 3-5 points; more might indicate a story that may not fit into a given sprint.

So what is wrong with this process? Nothing, if it works for you. It has never worked particularly well for me. As a tester, I have found acceptance criteria vague and largely useless to my testing efforts, making the task of test design a separate process to be performed after the story is placed into a sprint. That means I ask all pertinent, probing questions during that analysis process and the information I gather may or may not be known or shared with the rest of the team, most of whom are concentrating on development, blissfully unaware of the tests I'm going to execute against their code. That means they are not coding to pass those tests. I find this process very waterfall-like and non-collaborative.

In addition, whenever you introduce a constraint into a process, it becomes a potential problem. If more than 3-5 criteria mean something won't fit into a sprint and as a PO I really, really want this feature in the next sprint, I am likely to keep my criteria in the acceptable range whether it really belongs there or not. Wow; what a terrible statement, right? None of you know someone who has bent the rules a tad to get something they passionately feel needs to be a priority done ASAP? I've seen it often, in fact, I've seen it in every company I've ever worked with.

There is also a problem in terms of perspective. The acceptance criteria is supposed to act as gauge of "done". I have never found that to be particularly true. For example, acceptance criteria, written from a user perspective, rarely says "All my existing stuff still works after this change is implemented". Yet I am confident end users don't want a change to break everything else it touches. Perhaps a story cannot truly be estimated with any degree of accuracy without considering the size of the entire suite of tests that need to be run against the feature.

What I am proposing is that acceptance criteria be re-imagined into something more robust, realistic, and useful for creating quality software quickly. There is no point in producing software that meets user expectation, but cannot be maintained or breaks something else. The important and edifying information that comes naturally as part of the test design process needs to be driven out as early in the process as possible and the development staff need to know what their code needs to pass before they start their actual code development. I believe it is time to replace the acceptance test criteria concept with that of a collaborative test set model.

5 Why collaborate on testing?

Let's say that we are agreed, in principal, that a more robust definition of acceptance criteria might be in order. Why can't that be done by one person on the team? The PO, the developer, the BA, the SDET? It all has to do with perspective.

The developer arguably knows their own code better than anyone else on the team, and several development team members together might understand the technical aspects of a given feature better than anyone else. They might also have insight in terms of impact to associated or downstream applications that are not widely known by others on the team. Their perspective on what needs to be tested is unique because of their technical knowledge base.

There is, however, a phenomenon that is part of the human condition that tends to make us blind to our own errors. Consider that if I write a report and read it 10 times, I will likely find and correct many errors. The minute I pass it on to someone else, however, they find 5 more. Almost instantly. Some of the errors will be obvious. This is true of developers as well. It is difficult to find error in your own work. The prevailing theory is that we subconsciously do not want to find errors in our own work. In addition, development focus and training rarely centers around "how to find software errors", whereas much of a tester's entire career revolves around finding and reporting error. Books have been written, classes taught, and specific techniques implemented all specifically targeted at finding error in code. Where a QA or SDET resource tends to shine is in taking whatever information is available and specifically analyzing it and asking "what if?" to drive out error conditions or to validate error handling. The ability to not only recognize valid conditions (correct input yields correct output), but to ensure error conditions are handled gracefully, makes the perspective of the testing resource valuable to a collaborative testing effort.

A QA resource is not an end user. QA resources often push back at that statement, as they strongly feel they represent and support the end user. Not so. They may test like an end user, if their focus is on functional testing, but they cannot represent the end user – they simply do not understand end user workflows and constraints well enough to represent them. I would also argue that it is the development organization that is the primary client in terms of testing efforts. Testing provides information to development in order to help them meet their goals. The resource most likely to represent the end user is likely either the end user themselves, or the person who grooms the backlog or writes the user stories for the team. Most of us have experienced issues with a problem the team considered minor, only to find it had major impact on their user community. That makes the perspective of the end user critical to a good collaborative testing exercise.

These three perspectives form the triumvirate of what is minimally required to have a good collaborative test set. Does that mean no other perspectives are important? No. But this what is minimally required. When training others about collaborative testing, I always discuss the concept of *oracles*, which I define at a basic level as “anybody or anything that knows more about a given topic than you do”. I encourage inviting oracles to the collaboration process. That may be members of another team, a DBA, someone from devops, etc. Whomever you feel would have valuable testing ideas for the feature under discussion should be invited to participate.

When you collaborate on a test set containing multiple perspectives, your team aligns on what needs to be tested in order to pronounce a story done, you push test design to the start of a sprint, and you drive out anomalies in understanding early in the process.

6 Getting started

I started working on collaborative testing over 10 years ago after being inspired by a class on Just-In-Time Testing by Rob Sabourin. I adapted what I learned to fit into my environment at the time and that process has continued to evolve as agile methodologies matured and my employers and the teams with whom I worked became more sophisticated.

The basic concepts of collaborative testing are simple. Quality is the responsibility of the team. Testing is the responsibility of the team. In order to be effective and have a good testing effort, multiple perspectives need to be considered. Rather than operating like an old waterfall technique (build code, hand it over to QA), the testing effort needs to begin early in the process and the test, refactor, retest cycle needs to reiterate throughout the entire sprint. Decisions as to who tests what, when, and how are made by the team and negotiated early in the process with no single individual responsible for everything associated with the word test.

This differs from what is done on many agile teams in several important ways. First, test design is not a separate process done after a story is pulled into a sprint by one person, who is also going to be responsible for executing the test set. The test set is built collaboratively and replaces the acceptance criteria on a user story. This reinforces *quality is the responsibility of the team* and *testing is the responsibility of the team*. It ensures important perspectives are part of the base test set.

Collaborative testing consists of two parts. The first part is designing the collaborative test set. The second part is negotiating who does what.

Let’s start at the very beginning. If you work on an agile team, it’s likely you have a Product owner (PO) or Business Analyst (BA) in charge of a backlog. A backlog is a list of features and bugs that need to be coded. The PO will take of grooming the backlog, which is going through each item and determining priority based on user needs. Depending on your organization, the PO may target what sprint those items go into. Things can get a murky at this juncture, because the process used to size and estimate work varies from company to company. In my current company, once the work is prioritized and the PO has determined a desirable target sprint, a meeting is held to review the proposed work and design collaborative test sets (acceptance criteria). The work is then estimated and negotiation takes place as to what moves forward into the sprint. Collaborative test sets built for items that return to the backlog do not represent wasted effort, as those were high priority items that are likely to make it into a subsequent sprint, with the test set and estimation work already done. There is no longer a reason for any separate test design process; the base is complete. This saves time on the part of the tester. The development staff know what tests need to pass in order to consider a user story done – before they start writing code.

The entire team is aligned on what constitutes *done*. This sounds simple, but the concepts and actually making them happen can be a very complex issue for many agile organizations. Although agile methodology encourages testing early and often, and I don't believe I've ever read an agile treatise that did not recommend QA be involved at the story stage, actually making it so escapes many teams. They know it should be done, but don't know how to do that effectively.

The collaborative test sets are built using a modified Just-In-Time and lean coffee type of structuring brainstorming session.

7 Collaborative test sessions

The way I utilize what I learned about Just-In-Time testing over 10 years ago and the way I use it today are very different. I've simplified the process to make it easy for any participant to understand. The original process was taught to a bunch of people who were all testers. So they already understood invalid input, operational considerations, user-specific testing, etc. There were many categories of tests. I've reduced that to two. Valid input (which should yield valid output) and Invalid input (which should be handled gracefully by the system). What do you need to make it happen? A room, the proper attendees (which is anyone with a stake in the game, but minimally the PO, tester, and developer(s)), and a pile of sticky notes along with something to write with.

Collaborative test sessions are fun with the right group of people; very fast-paced and engaging. There are a few rules for the sessions, but just a few (in keeping with agile mindsets):

1. Everyone has to be prepared; if anyone has not read the user story or use case in advance, they are asked to leave (see "shunning" below)
2. The only person with a laptop is the person coordinating the session. This technique does not work effectively if people are on their phones or laptops – they are very focused sessions and require full engagement
3. There are no stupid ideas. If there is a test idea that does not make sense, it can be removed later on. The object is to participate and generate as many test ideas as possible in a short amount of time
4. The sessions can be timed (like lean coffee), but they cannot go over one hour without a break. If your stories are broken down to their smallest common denominator, the team may be able to build a test set in 10 or 15 minutes, which means they can get through 4-6 stories in an hour. Look at your throughput – how many stories does your team typically get done in a sprint? On our teams, it's typically 25 stories in a 2-week sprint. That means we put aside 4-6 hours during our planning to build collaborative test sets.
5. Anyone who is not prepared, does not participate, or sidetracks or torpedoes the session gets "shunned". That is, they are either asked to leave, or not invited to future sessions. The process is dependent on everyone being present in mind and body, focused on the task at hand. One of my peers, a development director, skyped me a few weeks ago to tell me he had been shunned from a collaborative test session for being unprepared. And his staff really enjoyed chivvying him from the room. We both laughed about it, but he learned from it and his team enjoyed themselves, felt empowered, and got down to business.

The session coordinator is usually the testing resource on the team, but this role can be assumed by anyone. It is up to the coordinator to invite pertinent players. Again, the players minimally include developers associated with the user story or use case, the PO or user representative, and the testing resource.

The session coordinator establishes wall space labeled VALID, INVALID and QUESTIONS. VALID scenarios are those where valid information is input with an expectation that some valid system response or transformation will occur. Often these are referred to as *happy path* tests. INVALID scenarios are those that validate error handling. Invalid input is handled in some graceful way, such as returning an error message. "QUESTIONS are things that come up that no one in the room understands or can explain in enough detail to formulate tests. If, as part of the user story or use case a bank of regression tests need to be executed, that is specified under VALID conditions. The statement might be “I need to verify all of the existing X functionality still works once this change is made”. There is no need to select tests or get into further detail during the session. The team builds out their test set by having the coordinator read off the user story or use case and generating test ideas. They do this by stating their test idea out loud, writing it on a sticky note, and placing it on the board as either a VALID test, an INVALID test, or a QUESTION. With a good, invested team this process is actually fun and little hectic. The idea behind stating the idea out loud is that one idea often spawns another idea. The coordinator ensures both valid and invalid cases are getting attention. For example, if there are many valid tests and only one invalid test, they might ask “What can we do to break this?”. It is normal, particularly from a tester’s perspective, to have many more invalid than valid tests. It is also up to the coordinator to ensure everyone is participating and that everyone’s voice is heard (“Hey, John, what do you think?”). There’s a difference between not participating and not being able to get a word in edgewise. The session ends when test ideas start trickling to a close; again, it’s up to the coordinator to call a halt when it makes sense. The coordinator then gathers the stickies, puts together a list, and sends it to the participants, asking them to review and update it (if they wish) by X date. Any questions generated by the team are also put on the list and the coordinator takes point in getting those questions answered and updating the team with answers.

This, then, represents a test base that contains a variety of perspectives and that forms the base for all testing to be done against that user story or use case. In my particular company, the testing resource stores the base in a centralized test management tool and translates the scenarios to test stubs that can be automated or arranged into logical charters for session-based test management. This also allows us to pull reports on demand and metrics as requested. Our test management tool interfaces with JIRA (a common agile task management tool) to further enhance management of testing functions.

This process takes the analysis tasks normally performed by the developer, test resource, and BA/PO and performs them in concert, collaboratively, driving out questions and establishing the test base before code is written. It adds minimal time up front, but that time is recovered (plus) as the sprint progresses. For agile shops that would like to move to TDD at some future date, this is a very good first step, as it establishes the habit of designing tests prior to writing code.

Companies heavily invested in innovation like brainstorming techniques such as this. At my current company, all of our agile teams, POs, and testing staff have been trained to use this process. It has been so popular and saved so much time that teams are now using the technique to collaboratively write user stories. Because all of the pertinent consumers of the user stories are together in the same room, all questions get identified or answered up front, allowing for better, more consistent information and common understanding of what needs to be developed and tested. Overall, this technique is just a quasi-structured brainstorming session.

8 Collaborative testing

To my mind, this is where the true nature of agile kicks in. The team, as one entity, negotiates who will test what and when. Normally, the nature of the tests themselves makes division of duties obvious. If, for example, I have a highly technical test that wants to validate a new service has registered itself and can be consumed, it’s likely that will be a unit test performed by a developer. An end-user would be unlikely to understand what that even means, let alone have the technical ability to verify it. In some cases, after division of duty has been made, the tester (whether developer, QA resource, or end user) will further drill down the tests in their pieces of the acceptance criteria until it makes sense for what they are doing. This is especially common in TDD shops. The developer may have a number of technical tests they want to code and run based on user story sub-tasks that need to be completed and that feed the test in the acceptance criteria. In fact, some TDD shops take their share of the acceptance criteria tests and run another collaborative test session with tech staff only to further drive out unit-specific tests. Remember that a collaborative test session can take as few as 10 minutes.

The testing resource, if an automation framework exists, can perform the same tasks as TDD development staff – that is, code their tests prior to actually receiving the code. In those agile organizations where testing staff perform all of the testing or automation tasks, their testing base is complete and they can proceed as normal.

The end user (or representative) now has a set of tests they agree constitutes *done* and adequately covers what they feel is acceptable – after all, they were part of the process. What does the term *done* actually mean? The definition of *done* is determined by the team. A simplistic definition would be code is complete, testing is complete, and the user has accepted it. So the user representative can either execute those tests they consider important themselves as code is made available, or ask to see test results from the team. This results in less thrash at the end of the process, as there are no surprises for the team as their user representative executes scenarios they missed.

In all cases, regardless of who does what, the definition of done is more concrete. Even in those environments that utilize solely manual test technique, analysis time is saved, the test base is more complete, and the development staff have the opportunity and advantage they need to produce higher quality code, simply through the expedient of understanding what tests will be run against that code.

9 What can you expect from this process?

Before you decide to invest time and effort into any new process, you have to be sold on the idea that it has some benefit to you, your coworkers, and your company as a whole. I'd like to talk about some of the benefits I've experienced and encourage you to experiment with collaborative testing yourselves.

I have had the great good fortune to work with fantastic testers and every team I've managed and worked with has kicked butt in terms of quality. Maybe not at first, but with hard work and through heavy collaboration and a willingness and strong desire to "raise the bar". It's up to you and your own company to determine what that bar is. It differs from company to company. Whether the team was manual testing only, automated testing only, or (more commonly) a combination thereof, the testing efforts have always been robust and the error rates in in the field have typically run at 3% or lower. That number comes from dividing the total number of errors found in the field by the total number of errors found during the sprint (or testing effort, if you don't use agile technique). In addition, our "measure of goodness" is that none of the errors found in production require immediate code changes or a "dot-one" (.1) release.

That number might tell you that I believe in and support metrics for process improvement and you would be right. Metrics, pulled the right way and for the right reasons, can tell a story. Whenever we've implemented and tried a new technique to see if we could raise our own quality bar, we have had common metrics to help us figure out if we're being successful or need to try something else.

With collaborative testing, since we were using some of the time and efforts of other teams outside our control, it was particularly important to management staff to determine if collaborative testing was a good investment of time and resources (which equates to money). What we found was this:

1. In cases where the error rate was already gratifyingly low in production, the primary benefit of collaborative testing was the savings in test design and execution time. We were able to cut 3-week sprints to 2-week sprints with no loss in production quality.
2. In companies without a robust testing process, using collaborative testing decreased the error rates in production by up to 20%.
3. In every case, whether robust or not, regardless of project methodologies, the error rates of code submitted by development staff decreased by up to 40% or more. That means using the same measure as above, we measured the number of errors found in the QA environment divided by the total number of errors found during the project life of the product under test. In the case of one team, it decreased from 400% (an average of 4 errors per test were found) to 20%. Within 6 weeks. This has been a huge win in every company I've introduced to this technique; the lower error rates allow for code to move through the testing process much more quickly and involve much less development time for coding fixes. In every case, it allowed the agile teams to condense and create working code ready for production in less time.

4. User sign-off has been obtained more quickly. Although agile process requires a definition of done up front, many companies flounder with this, particularly if their test set is not part of the user story contract (e.g. acceptance criteria). Acceptance criteria can pass and the product might not be ready for prime time (such as breaking major features elsewhere, negatively impacting response time, etc.). I worked for one firm where getting an approval from the user organization took months. The collaborative process reduced that to days, and the user organization chose to stop testing altogether (which saved them time and money), merely asking to be copied on test results. Having a more definitive definition of *done* helps the entire team focus on making that happen.
5. It allows development staff an opportunity to morph into TDD more painlessly. In my current company, we started with collaborative test tests and have had two teams move into TDD with relatively few hiccups and two more that are anxious to join the parade.
6. Teams start to use collaborative techniques for other efforts. The agile teams I work with have found value in the targeted brainstorming learned through collaborative testing and have applied it to many of their other tasks – from forming temporary SWAT teams to address specific problems, to documenting user stories, to coming up with discussion points for innovation.

10 Learning more

I can't refer you to any books; there aren't any on this subject. But I've included a few websites that address brainstorming technique if you've never tried it. The way I learn is to pick a team, a place, and a time, and experiment. This particular set of experiments started over 10 years ago.

I always start collaborative test efforts by training all of the staff. All of our agile team members have been trained in this technique. I try to make that fun and very "outside the box", and it's been very successful. For example, in late November, I took what is arguably one of the Worst Movies Ever Made – Santa Claus vs the Martians, and made one of the premises into an epic. That is, "I, as Santa Claus, want to deliver toys to the Martians". It was up to the team to brainstorm on what that would entail. I let them choose their own coordinator and halfway into the process, visited them as an oracle – Santa Claus – to answer any questions. A lot of candy canes were flung about. Overall, it was fun and the team learned to use the process in a painless, non-threatening way. The test sets were pages long (which was great, as it showed the team how much can be covered in a short time) and we were all entertained reading the collaborative criteria results.

Silly as that exercise was, everyone learned what they needed to learn and the results have been everything we had hoped and more. The nature of our work requires change and constant improvement. I hope the potential and possibilities of this technique resonate with you and you do some experimenting of your own!

.

References

Web Sites:

Lean Coffee Organization. http://LeanCoffee.org (accessed July 21, 2016)

Brainstorming methods. http://mindtools.com (accessed July 21, 2016)

Information on testing biases: Kilby, Richard Aug 27 2014. Paragon blog Aug 29, 2014, How We Miss Bugs.
https://www.paragon-inc.com/resources/blogs-posts/~/media/Files/How%20We%20Miss%20Bugs.ashx
(accessed July 21, 2016)

Rob Sabourin classes: http://lets-test.com/wp-content/uploads/2011/10/2012_05_06_Just_In_Time_Testing.pdf