QUALITY OWNERSHIP

Linda Wilkinson

wilkinsonlinda4@gmail.com

Abstract

While technology has changed significantly and SDLCs (Software Development Lifecycles) overall have morphed with the advent of agile methodologies, quality processes have not kept pace. The problems we had 40 years ago are problems we still deal with every day. Why are we doing this to ourselves? Our standard processes didn't work very well 40 years ago and they don't work very well now.

Quality Ownership is designed to take agile processes to a new level using shift left thinking prior to making tickets ready to work, using the expertise of the Quality Analyst to manage the quality landscape, the SDET staff to begin automation early, and the agile team itself to assume responsibility for quality as a team. Givelify has experimented with this methodology, we've learned a lot, and our results are exciting enough we would like to share them with the rest of the quality community. Our Quality Analysts are able to serve multiple teams (at a savings of hundreds of thousands of dollars), our quality levels in Production remain high, and our agile teams are better able to absorb last-minute changes to support our business.

Biography

Linda Wilkinson has been a QA practitioner for 40 years (!). She started her career in software analysis, moved into QA, and followed the traditional route of QA Analyst I, II, II, Senior, SDET, Lead, Manager, Director, VP, and took a few brief forays into DevOps, DBA, Product Management, and data analysis as it became necessary in order to accomplish quality goals. At last count, she has worked with 18 languages, helped author a book, and has had the privilege of speaking at a variety of conferences, both as a speaker and as a member of expert boards in QA and metrics. She has built QA teams (functional, automated, and performance) from scratch five times. Passionate about quality and providing business value, her teams have maintained error rates close to 1% with high customer satisfaction ratings in a highly competitive and ever-changing technical landscape.

Formatted: Font: Italic

1 Introduction

In order to understand the ideas behind Quality Ownership, a basic understanding of the history of the field might be in order. This is a brief overview of the quality world during the past 40 years or so, told from the perspective of a practitioner.

40+ years ago, everyone was using Waterfall methodology. Many modern perceptions of what waterfall was like are, well, just wrong. Final implementation to Production might have been "Big Bang", but large projects were broken down into phases, often a month long. You might compare it to a 4-week sprint. At that time, in the author's world, QA staff were developers who reviewed code, made structural updates, ran tests, and fixed bugs. This was done at the end of the phase and again at the end of the entire project. That final review/test phase could bog down for months. QA was considered a bottleneck. The mission of the QA team was considered a failure if errors were detected in Production.

The QA world changed in the early 80s, when a book that had been around since 1973 started getting some traction – Glenford Meyers' "The Art of Software Testing". It talked, for the first time, about the psychology of finding errors; pointing out realities that still ring true today. For example, it talked about WHY it's difficult to find errors in your own code. Consider, for example, that I write a report for the CEO. I want it to be well-received. I may read over and check my work ten times. I will find and correct some errors. If I then ask a colleague to read it just as a final check, they will find multiple things I missed – some of them painfully obvious. Why did I miss them? The reason is psychological. We don't really want to find errors in our own stuff. In essence, developers test to prove their code works. QA Analysts test to prove it doesn't. Just that difference in perspective drives out error. But for me the biggest take-away was unless you are writing code for other developers, your clients do not care what language your code is written in. They don't care about your environment, your methodologies, or your expenses. YOU care about those things. They just want something that does what they want it to do and doesn't break. Now there's a simple concept of "quality", although it's incomplete (we'll talk about that later). So the field morphed and QA as a whole started testing systems from a user perspective. While this certainly was faster, responsibility for structure, bug fixes, and everything but the actual testing devolved to the development staff. Tests were defined by QA as soon as the analysis document was available, which took time, and as code that was often untested and incomplete was passed over to QA staff for testing at the end of a phase or project, the QA process overall was still time-consuming. QA was still considered a bottleneck. Errors in Production were still considered a failure on the part of the QA team.

Enter Agile. Agile methodologies came out in 2001. What is really interesting about Agile is that it's 22 years old and everybody is Agile. Or are they? Many of the original precepts behind success for the original agile teams have simply been lost as our field continues to evolve. Do you have a closely-knit, co-located team in the same time zone? Provide food? Have someone that protects the team and does not allow them to be interrupted? Do they pitch in and help each other out? So they're all full-stack developers? Does the entire team do the testing? If not, you may have something, and it might work well for you, but you're not really close to the original concepts behind Agile. And what about QA? How did their world change? Here's the bad news. It didn't. It's just in 2 week chunks instead of 4 week chunks. With the advent of exploratory testing, which most companies still think means "no documentation" or preparation, our terminology changed, but the methods we use to uncover and report bugs really didn't change. If there is a QA person on the team, they are likely responsible for testing, regardless of when or how things are delivered to them. Chances are also good QA will be responsible for test automation, and if something fails in Production, it is likely the QA team getting the call; "Why did you miss this?". In a truly Agile shop, this would be viewed as an agile process error; everyone is responsible for testing and the quality of the project. But in reality, QA is still considered a bottleneck when the team fails to make a deadline and a failure if errors manifest in Production.

There have been and are many flavors of project methodology, but the mission and methodologies of a quality professional have not changed significantly since the early 1980s. RUP (Rational Unified Process), MBO (Management by Objective), Six Sigma, Lean, Scrum, Kanban, TDD (Test-Driven Development), BDD (Behavior-Driven Development) and all derivatives thereof – it doesn't matter; a QA professional needs to be able to support any type of project methodology and provide value to their teams

and their company. Our job is to adapt. From process-heavy methodologies like RUP, to admin-heavy processes like MBO or Six Sigma, to the ceremonies and time-boxed Scrum methodologies and the C/CD (Continuous Integratipn/Continuous Deployment)-friendly looser weave of Kanban, all of these methodologies can and have been successful. Methodology of any type can and has been unsuccessful as well. Overall, success of any given team is more about the team itself. A talented, invested, dedicated, supportive and supported team is going to make things work in spite of obstacles. So why work on methodologies? Because we can and do hamstring our own teams, burn them out, and de-motivate/demoralize the very people we depend upon.

2 Realities

Let's consider the nature of testing itself. While a test can be developed before anything actually exists to run it against, the testing itself cannot be performed and give you any meaningful information until has something to test. "Test early, fail often" sounds great and all testing is valuable, but the errors found as you shift left (test whatever you can as early as you can) are different than those you find later in your development cycle prior to production. My code may be tested and completely performant, yet the nanosecond I put it in place and it interacts with other code in a more production-like environment, it can fail. It can cause other code far down the line from my own to fail. Overall, it is the later phases of testing that are most indicative of performance in production. It is time, after more than 40 years, to recognize a primary truth about testing. It takes place at the end of something. It takes place once something is done. The most meaningful testing takes place at the end of a development effort when all of the pieces are ready to interact with each other in an environment as close to production reality as possible. The reason testing extends a deadline is that its very nature demands end-of-cycle execution. The purpose of testing is to find errors, all of which have to be assessed and potentially fixed by the team. This takes time and effort by human beings. It is true whether the tests are automated or manual as results must be analyzed, assessed and potentially fixed by a human.

This testing reality is true regardless of methodology. So, the first challenge in terms of change for the better is finding ways to make an activity that can't take place until something is done faster without losing any of the benefits of that activity.

Diving even deeper, what is "testing"? Testing is comparing a given result against an expected or desired result and noting anomalies so they can be repaired – either prior to Production or later. Or the decision may be made that the anomaly is acceptable as it stands. How is this decided? In other words, who decides what is good? From a practical perspective, eventually it will be your customers that decide what is good. But using our simplistic definition above, your product does what is expected and doesn't break; the expectations against which to compare the results of your testing is documented somewhere. In Waterfall days, these expectations of what needed to be produced, what it should look like, and how it should operate were in a bulky analysis document. This document took a very long time to prepare, but it did have some advantages. First, everyone on the team had a complete picture of what was desired before coding or testing started. Second, it was so cumbersome to create, changes were discouraged and normally had to go through some thorough vetting and a change process before it was accepted. This was both a blessing and a curse. The blessing was that spurious and unnecessary changes and bells and whistles were not added as often, which kept disruptions to the forward momentum of the team(s) to a minimum. It was a curse in that the teams were far less responsive to changes in the marketplace and technical issues or discoveries later in the process caused longer and more stressful updates to what had already been done, with the added burden of ensuring everyone on every team was kept in the loop about those changes.

Now most analysis work is kept in a variety of mediums. There might be one site for designs and the work is likely broken down into pieces ("tickets") in a project management tool such as Jira. Jira tickets and similar contemporary project management tools are also both a blessing and a curse. The blessing is that a ticket represents one small element of work and is fast and easy to produce. Changes are easier to absorb. While a change might have to be vetted, it might be by a single team and only to determine if it

can be handled during the current sprint or needs to be placed in a later sprint. There are a number of curses. Reading a single ticket does not necessarily give a very good picture of the overall effort. Changes may not be vetted as extensively for impact to the product and team or even if it is desirable from a user perspective because the ”approving body” is much smaller and far more focused on their own specialized tasks. It is also unusual for a ticket to be detailed enough to use effectively as a basis for testing. Tickets are “supposed” to contain acceptance criteria. Often they do not. They may contain generic statements that are a re-iteration of the title or intent of that piece of work, but acceptance criteria go far beyond that in reality. For example, if you are creating a log-in screen, your acceptance criteria might simply state the user is able to log-in to your application. A testing professional is going to have quite a few tests associated with such a ticket. What happens if the log-in is wrong? If an incorrect log-in allows access to your application, that anomaly, or bug, will be written up and the team will have to address it. In other words, the acceptance criteria is actually every test which if failed will result in a critical bug which must be addressed by the team.

So who is responsible for determining the test set for the ticket? The product owner, manager, or BA might be responsible for creating initial acceptance criteria. But it is normally the quality professional who is responsible for taking any design documentation and the ticket and determining what needs to be tested. If the QA personnel on your teams do not begin this work until a ticket is pulled into a sprint to be worked, or they do not document what they intend to examine, your agile team is already behind. The critical analysis piece of the equation will be started at the same time as development. There is no way they will be able to use those tests for their own unit/shift-left testing, and getting automation done prior to the end of the sprint will be problematic.

Ultimately, the test analysis is the most critical part of the testing process. It doesn’t matter what kind of analysis documentation you have (or don’t have). It doesn’t matter what kind of project methodology you use. The key to ensuring a valuable testing effort is running the right test, at the right time, in the right environment and providing the results to the team. Shocking as it may seem, it makes no difference whether that test is manual or automated. It doesn’t matter who actually executes the test. The key is running the right test, at the right time, in the right environment.

This means the most important part of the entire testing process is test analysis. Determining what needs to be examined and comparing actual results to expected/desired results. Good testing is always a comparison. There is a big difference between executing code and verifying code. When there are no expected results, it is not possible to ascertain whether a result is desirable or undesirable except through the experience and expectations of the tester themselves. This will work well if your tester is highly experienced, has extensive product knowledge from your users’ perspective, and is 100% available. They will make good guesses most of the time, although a chunk of time is going to be spent consulting with others to get their perspective on any perceived anomalies. But it represents a risk to your company and is ultimately not sustainable. Most companies would not want the future of their organizations dependent on guesses, yet many operate this way with varying degrees of success or lack thereof. It is also important to the team that this test analysis is done as early in the process as possible. This allows development staff to understand and fold testing expectations into their work and the entire team to agree on what “done” looks like.

Thus far, we’ve determined that testing takes place when something is done and that the key to successful testing is good test analysis done early. Let’s talk about people next.

3 The Disconnect

There are many myths around the testing practice. Everyone who has ever tested anything, including a battery, believes they know everything about testing, that it’s easy, and that the entire field is somehow a lesser activity than development. Good testing requires training, experience, and discipline. Testing is part of the development process. The specialized nature of the field should be evident; there is a reason QA professionals find so many issues after development is complete. Why aren’t developers catching these errors? Where is the disconnect? Is that perception that’s been around for 40+ years that “developers are terrible testers” really true?

The short answer is no. But let's talk about the disconnect. Say my company is an on-line marketplace. We add vendors to our system, which enables them to make their goods available to customers who can purchase said items on-line. A QA professional is going to know how to add a vendor, add goods, access those goods as a customer, purchase goods using a variety of payment methods, and verify the vendor and my company got paid. They may also check any analysis/reports provided to the vendor or receipts to the customer. They are going to know these things and more because they have to in order to do their jobs.

A developer, on the other hand, is likely to specialize in either front-end or back-end work and be deeply, technically involved in one small part of that chain. Generally, people learn and do what is required for them to be effective and no more; most people don't have the time (especially in the IT field) even if they have the interest. There are developers out there doing good work and contributing to their companies that have never even logged on to their own systems as an end user. Does that mean they are incapable of learning? No. But it does mean they lack the breadth of understanding of their own systems from a user perspective necessary to do good test analysis. Think about the best developers you've ever known. You know the ones – you call them at 3 AM when something goes awry, whether it's actually in their scope of responsibility or not. It is highly likely the development staff most valuable to you are those that know your system from BOTH a technical and a user perspective. A customer support person can explain what a user did and they understand instantly and ask questions customer support understands, then can pinpoint the area of code most likely at fault and engage their cohorts in finding a solution. This level of employee is going to be better at test analysis – unfortunately they are rare and unlikely to be used for such a purpose.

What this means is the disconnect is due to the learning requirements and time available to gain the expertise necessary to be an effective test analyst and not capability. It is again the test analysis part of the equation here, not the actual testing. The actual testing activity does not require any particular skill. It can, after all, be run by a machine. Anyone with time available and understanding of basic skills can run a defined test. Understanding what test needs to be run? Test analysis is a highly specialized skill.

What about other members of the team or the company? What about Product Owners? Product Owners certainly understand the system from a user perspective. What they lack is training and time. Does your Product Owner/Manager have the right skills to perform test analysis? Take a look at their tickets. What does their acceptance criteria look like? Do they provide any? Do they include how the system handles invalid or unexpected input? Do they mention downstream applications that need to be verified as part of the change? If not, even your lowest-level Quality Analyst is going to be more effective at test analysis.

What about an SDET (Software Development Engineer in Test)? This is an interesting question because there are so many assumptions and myths surrounding test automation as a whole. There's an assumption that test automation is a recent development; it's actually been around for at least 35 years. Every company, everywhere, always wants their test bank(s) automated. And why not? Something that takes manual testers weeks to execute can take less than a day, even with the human interaction portion of reviewing test results and failures. But in order to automate a test bank, you have to have a test bank. And again, who is going to design your tests? An SDET? They just want to automate tests. They are developers, not test analysts. They interact with, think like, and work like development staff. They tend to miss the same bugs development staff do. That makes sense, because their understanding of your systems from a user perspective is likely on the same level as your development staff. A common myth in the IT field is that an SDET is just like a Quality Analyst, except they can also automate tests and are therefore a better investment long-term. Unfortunately this is just not true. Even if it was, an SDET with years of experience is going to be expensive, and generally they will not enjoy and do not want to be involved with any significant tasks that do not involve automation specifically.

Where are we now? Thus far, we've determined that testing takes place when something is done, that the key to successful testing is good test analysis done early, and that Quality Analysts are best positioned to perform test analysis. The last reality to tackle is process.

4 Opportunities for Change

It is to be hoped that any company interested in innovation and with some claims to operating in an agile environment periodically review its own processes, identifies those that are holding it back, and either provides training or tosses them accordingly. That is, unfortunately, a hope in vain. People resist change. There is some comfort in familiarity, and fear of losing respect by not being expert (or being unable to be competent) in something that is new. In addition, there is a warm, fuzzy feeling when you are part of a pack or you do things the same way these other 2000 reputable, successful companies do. That is not, however, necessarily in your best interest. Instead of being a follower, you could be a leader. And leaders do what is in the best interests of their own company. Not someone else's. We need to look at our processes and be tenacious about cutting what doesn't work.

Let's look at a few agile processes for a moment. If you do not have a tight, co-located team in the same time zone, are you agile? If the answer is no and you're successful, who cares? You can call your process anything you like and laugh all the way to the bank. If the answer is no and you're struggling, with processes that do not work for you, it's time to take a second look. Are all your team members in the stand-ups or are some in a time zone that does not permit that? How do you handle that problem? Is it really handled, or are some team members unavailable or not in the loop? Are your retrospectives more than a 4-syllable word? What valuable changes have you made in the past 6 months due to retrospectives? The idea of a retrospective is great. Lots of ideas are great. But unless you appreciably benefit from an agile ceremony, let it go. What about RCAs (Root Cause Analysis)? Do you do them? Are they about improving processes and tools or providing training or capability growth, or do they simply blame people? Do your people embrace them, or do they dread them? If the answer is b., what are you going to do about it? And if you've been doing these same ineffective, damaging things for years, then just stop. Achieve your goals in some other way.

With this in mind, let's look at a typical scrum-like project sprint. A change or new feature is identified. Any new/changed screens and workflow are worked on and placed in (some) tool. The workflow and changes are broken down and estimated by a team(s). Including QA time in estimates may or may not be included. The Product Owner pulls tickets into the next sprint based on team capacity. Team capacity may be based on historical data or may not. The development staff starts working. The QA staff begin test analysis. Once an individual ticket is complete, it may be tested by either the developer or by QA in a non-prod-like environment. Bugs are written and the fix/retest cycle continues. Once all the tickets are complete or a working feature is available, the QA staff retest the entire feature in a prod-like environment. All needed tickets may not be available at the same time, resulting in some delays. In addition, several design changes might take place coming directly from the executive team. Errors in the estimation process (perhaps using story pointing) become evident, but the team is committed now and moves forward. The fix/retest cycle at this point in the process strongly impacts team delivery dates. Once the delivery date has been reached, the team schedules its next sprint, with QA staff taking time out of their testing (and developers of their fixes/resubmissions) and development begins for the next sprint, although the last one is not complete and in production. If enough time passes during the final fix/retest cycle, two sprints may be combined. Team capacity may or may not be updated accordingly. Once the sprint is in Production, tests are passed off to SDET personnel for regression automation.

Does any of this resonate with you? This process is not effective. We need to address when test analysis takes place, when a ticket is pulled in as "ready to work", how tickets are estimated, when tickets are automated, who is responsible for the physical activity of testing, and how team capacity is determined. Note there was no mention of addressing design changes coming from the executive team. If this is part of your reality, a better perspective is to ensure your final processes support these as a "business as usual" part of your landscape. Admonishing an executive team is not an effective strategy.

Before we continue with a potential solution, the above examples are IT-related, but there needs to be some discussion regarding quality as a whole. While the most simplistic definition of quality is that it does what it is supposed to do and does not break, there is actually more to the concept of "quality" than those two items. It is not a matter of who has the prettiest website either, although doing your homework and understanding what your customers want is important or no one will buy what you're selling. But when

someone thinks about quality and whether your company has it and how they "grade" you on a variety of sites is dependent on absolutely everything that touches them. You can have a killer app that has zero errors, but if your customer service rep treats someone poorly or cannot solve a problem, you've lost a customer. If you have any competition whatsoever, you have to pay attention to and "test" everything that touches your customer. Your website. Your social media posts. Definitely your customer service practices. Emails. Texts. Mailings. Everything that touches your customer influences their perception of the quality of your organization. This is a basic reality. Even if your customer base is locked in and has no choice but to use your apps, that could change in the future – best to treat them like the gold they are right now.

In summary, we now know that testing takes place when something is done, that the key to successful testing is good test analysis done early by the Quality Analyst, and that our process is broken and needs work. We would also like our solution to be useful to any part of our company that touches our customers. Let's talk about a potential solution to these problems.

5 Quality Ownership

There was a conference for quality professionals back in the 1980s called "Quality is Free". Quality is not, and never has been free. Quality is expensive, both in time and resources. It is, and always has been, difficult to justify and obtain sufficient quality professionals to handle the workload. The only way to support growing applications and company size is to expand the QA presence accordingly. What if we took a page from the book of other specialists on agile teams and made it possible for a single Quality Analyst to support more than one team? DevOps personnel, for example, might support multiple teams, as can Product Owners, UI Analysts, and others. What would that look like?

First, the single most critical contribution a Quality Analyst makes to a product team is establishing the test base. That is a task that others on the team cannot perform as successfully (see Realities above). If we agree Quality Analysts should establish the test base, when and how should that happen? If we want to solve the problems of not having tests available to be incorporated into shift-left testing efforts, having tests automated by the end of the sprint, and development beginning their work potentially weeks before QA begins theirs, it makes sense to say the Quality Analyst will establish a test base prior to a given ticket being pulled into a sprint as "ready to work". A separate paper could be written just to deal with the "hows", and it's highly dependent on how a company operates now, but this would mean involving the QA Analyst during the design phase, ensuring tests are documented, stored, and attached to the ticket prior being pulled into a sprint. I use the word "sprint", but if you work with another methodology, it would be prior to whatever block of time you divide into work chunks.

Once the development staff pick up a ticket to work, there are two possible scenarios. The original idea was to hand the ticket over to an SDET at the same time the developer pulled it in to begin work, but then an interesting conversation took place with Viktor Mateleshka of Givelify. Viktor had a team with no QA resources on it, producing some work for internal customers. To be frank, the QA team, in particular the QA Analysts, were a bit chary of Viktor; they felt he didn't recognize their value and were afraid he wanted to get rid of QA altogether. In actuality, Viktor had a high degree of respect for the Quality Analysts and credited them with a great deal of value in designing tests and finding critical bugs. What Viktor felt was overkill were the SDET staff. That is frankly an astounding, ground-breaking, non-typical point of view. What he said was that he had a whole team of developers. With an established automation framework, any one of them could automate a test. What he did not have, and wanted, was all of the experience and understanding of the systems from a user perspective helping his teams determine what needed to be tested. In other words, he wanted test analysis. From there, he felt his team(s) could take over. Regardless of whether this task is assumed by an SDET or a developer, tests can be developed before code is available with most contemporary automation frameworks. That is one of the foundations of TDD (test-driven development). Once elements/objects and actions/functions are defined, these are easy to build in a basic form, although more tweaking may be necessary as the code develops and the tests are actually run. As part of the development efforts, time to develop the tests would necessitate considering

them as part of the estimation process, improving the track record of the team for on-time deliveries. In addition, the availability and running of automated tests, incorporating them into CI/CD processes, would benefit the entire team in terms of time. It would encourage teams to automate, and automate early, which is very much in keeping with the direction in which agile development is moving.

The most critical and controversial part of Quality Ownerships is the question of who actually performs the testing. In short, anyone on the team other than the Quality Analyst. This is a non-negotiable key point to success for the entire Quality Ownership process. Why? Because breaking that "QA will test it" mentality will not go away and the teams will not grow and move forward until those crutches are no longer available. I saw a podcast recently by Elisabeth Hendrickson that talked about a situation she had as a QA leader where her team was being blamed for a production failure and she had no people to spare to help them out any further. The development leader was furious. But reality is reality, and his team was forced to carry on with no testing support. Guess what? They became better. No amount of QA professionals on the team had improved the quality of their output, but once they were responsible and assumed responsibility for the testing, they were forced to improve.

The idea of expecting the team to test their code also forces other staff to learn how to use their own product from a user perspective, which grows highly expert, valuable personnel. The available test pool grows from one person, often trying to "save" the team by testing overtime at the last minute, to the entire team, all working together to get the job done. That is the epitome of agile methodology. Again, with testing genuinely part of the development process, estimation techniques would necessarily have to improve in order to allow for the time to execute the test/retest cycle. And those valuable, expensive QA resources can support more than one team. If their primary job is to identify the test set, once that is complete they can move over and help do the same thing for another team.

How does this relate to non-IT teams like Customer Support, Marketing, and others? Anything that touches a customer should be "tested" prior to presentation to that customer. This would be a simplified process; a Quality leader would identify what needs to be verified and ensure those things are done and acceptable prior to presentation to actual customers. It could be as simple as establishing a process whereby all blog posts are reviewed prior to posting. It could be as complex as a Marketing Campaign, with all that entails.

In the next section, we'll talk about actual implementation and early results from experimenting with Quality Ownership on selected projects.

6 Implementing Quality Ownership

Change is hard and this one is a radical departure from what makes your teams comfortable. Those of you who have had to shepherd a team through any type of radical methodology change will understand immediately what types of challenges you'll face when trying to implement Quality Ownership.

First the good news. Givelify is a particularly good place to experiment with new things and I would like to acknowledge their participation and support. They are a small, (slightly over 100 employees) company dedicated to "the best of the best", which is likely why they are #1 in their particular niche, which is enabling charitable donations through an elegant set of phone apps that their customers love. They have visionary leadership and a particularly strong interest in both innovation and education. They are interested in Quality Ownership and agreed to experiment with it with one of their programs (3 teams). Those 3 teams are currently utilizing 2 Quality Analysts and 1 SDET. Since implementation, both Quality analysts are supporting more than one team, the SDET supports 3, and quality levels in Production have not declined. The quality levels at Givelify were very high to begin with (their error rate is under 1%), but what is important here is that the changes underway have not negatively impacted their client base. It is the intent of the company to continue what they've started and work with the new processes, adding additional teams as they gain in comfort and confidence. The company has been working with Quality Ownership for only a few months, but code delivery is no longer dependent on the availability and findings

of one individual (improved delivery), headcount has not been increased (6 were planned), and quality has remained high. Further assessment of savings and benefits/problems will become evident over time.

Atlassian, a 95+ billion dollar company, has done a great deal of interesting and brilliant work in regards to what they refer to as Quality Assistance; a link is provided below and will give you an idea as to the scope of this type of change. Where the techniques in this document differ is in the role of the quality expert in the process. Atlassian utilizes their quality staff as mentors and teachers with responsibility for making the teams better. The Quality Ownership concept includes the quality expert as part of the contributor to the team in a different but complimentary way. Part of the reason for that is purely due to human nature and the difficulty of change. A team member that produces a product, just like every other member of the team, is viewed differently than a member that proffers advice and "doesn't get their hands dirty".

It would be a disservice to you and to the field to not address the challenges and problems with implementing Quality Ownership. This is not a trivial change. The first problem has to do with Quality Analysts and their view of their work. Many Quality Analysts identify and define themselves through actual testing activity. While having to perform the work under a constant cloud of pressure is not especially enjoyable, testing, in and of itself, is fun. Quality Analysts are accustomed to being the one to find the errors, work overtime to save their team, and generally act as corporate heroes, even if just within their own frame of reference. That's hard to give up. What was found during implementation of Quality Ownership was that finding some conditions under which Quality Analysts could physically test, such as cross-team efforts (especially across programs), added to their satisfaction levels. In addition, the field continues to morph into an ever-increasing world of test automation and asking your analysts if they want to remain relevant in that world will provoke some good conversations, thought, and a willingness to try it out.

On the development side, developers are going to kick back against this process. They are already tremendously busy, and they definitely do not want to add testing to the mix. Particularly if that testing includes manual testing. That's actually a good thing, as part of the object of the methodology is to encourage test automation. After experiencing the changes for several sprints, a few major points came up that need to be considered. The Quality Analyst cannot be available to assist with the physical testing activity. If they are, the team will react by reverting to the way they always did things. There is a tendency for back-end developers to accept the methodology more easily than their front-end counterparts. This is likely because they are more accustomed to doing at least a portion of their own testing. Depending on the company and the skill sets of their quality staff, they may or may not be comfortable with "headless" testing - that is testing without a front-end feeding in data. This type of testing requires use of a tool such as Postman or SOAP/SOAPUI. It's possible back-end developers in your company do their own testing. Front-end development staff, however, will have to become familiar with some sort of testing architecture and build their tests accordingly. This activity would be required for any type of automated or TDD/BDD methodologies, but it is human nature to resist change and busy staff members will push back even harder against learning or doing something new. There needs to be no other options. Your SDET personnel can help here both in setting up the architecture, assisting the development with automated test development, or assuming responsibility for test automation, as best suits your company situation.

It is strongly recommended that when first implementing this process that some random checking be done to ensure the tests identified were executed and verified. It may seem laughable to some (a painful reality for many QA personnel) that a developer may not understand the need to test certain areas and will therefore opt not to run given tests. When this process was begun, that very scenario arose and the random check uncovered a critical bug. You will need some method to ensure all identified tests are run and results evaluated.

Management and executive management have to hang tough. It is very hard not to cave when your teams tell you they can't handle testing. If testing is simply not available, the teams have no choice. In these cases, reminders that there are companies out there with no QA staff that are doing just fine, that the QA staff are still available to answer questions, and pointing out the advantages to adding these skills to their toolbelt can be helpful to their careers. It has the advantage of being true. Ultimately, you cannot

move your company forward with this or any other methodology if you, as management leaders, allow individuals to undermine your efforts and thus the progress of your company. Good training, patience, and understanding are all going to be required.

Once the teams become accustomed to performing testing as a team and accepting responsibility for the quality of their code as a team, there is no reason the “no testing by Quality Analysts” rule cannot be lifted. But we need to move quality from “that person’s responsibility” to “our responsibility”.

Over time, assessment needs to be done to determine if the new methodology has had the positive impact overall originally expected. Have the team members become more expert in using the systems they build from a user perspective? Does the full team pitching in to get the testing done cut down on how long it takes to complete the fix/retest cycles? Are Quality Analysts able to handle more than one team? Have estimates improved? Is test automation done more quickly and is it being used?

Time will tell. As a personal note, I encourage all of you to experiment with this process and go change our quality world. It’s about time, don’t you think?

References

Myers, Glenford J (2006). The Art of Software Testing. Wiley India Pvt. Limited

Craske, A. (2021). How Atlassian Does Quality Assistance

https://qeunit.com

Hendrickson, Elisabeth (2023). Achieving Better Outcomes Through Structure

IT Revolution Episode 3, https://itrevolution.com