

THE COMPLETE GUIDE TO

SHIFT LEFT
TESTING

The whys and hows of
pioneering the movement

INTRODUCTION

In an Agile world, software and IT teams are under constant pressure to move faster. Typically, this means decreasing the relative length of delivery time while continuing to improve quality on each successive release. At the same time, there's always pressure to minimize testing costs.

Many organizations who have adopted agile development initiatives have focused on shortening sprints and incorporating customer feedback into features faster. But soon they run into severe quality issues, and their customers end up suffering. An emphasis on testing and quality has

made a huge emergence in the software market, where "move fast and break things" is no longer viable. A quality assurance movement known as "shift-left testing" has come along in response to this mantra.

This e-book will cover 1) why you won't survive if you don't shift, 2) how your testing processes should change when you adopt a shift-left testing methodology, 3) who needs to be involved in this movement and how, and lastly, 4) we'll hear from individuals about how they pioneered the shift in their teams.

TABLE OF CONTENTS

THE NEGATIVE EFFECTS ON CULTURE

THE HIDDEN COSTS

THE BENEFITS OF SHIFT-LEFT TESTING

SHIFT-LEFT TESTING RESHAPES PRODUCT DEVELOPMENT

Design Phase: Testing feature ideas

Using tests continually to guide development

Designing test plans

Test early and often, automate early and often

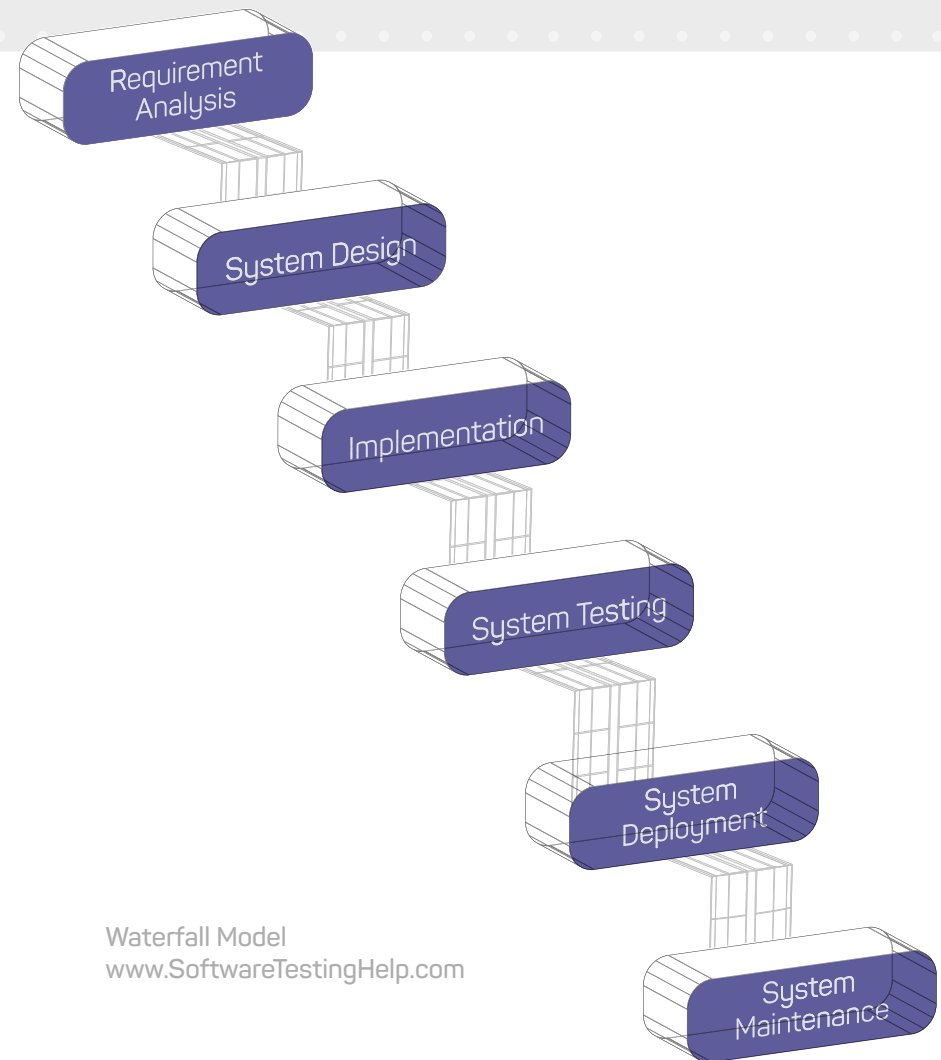
It's not just about testers

OTHERS HAVE MADE THE SHIFT - AND YOU CAN TOO

RESOURCES FOR CONTINUED LEARNING

THE NEGATIVE EFFECT ON CULTURE

In the past, most software teams employed a waterfall methodology. The workflow for these teams resembles a staircase. In this conventional approach to development, only the product team works with customers. They eventually return to internal teams with user requirements, a roadmap, and an aggressive schedule. Then, a short meeting occurs at which developers walkthrough the new feature proposals. Tasks are assigned and everyone goes off to their silos to work. If any testers get access to these early stages, it's typically limited to observing the developer planning session. With little involvement up to that point, QA is then asked to give effort/duration estimates on what it will take to test the new code.



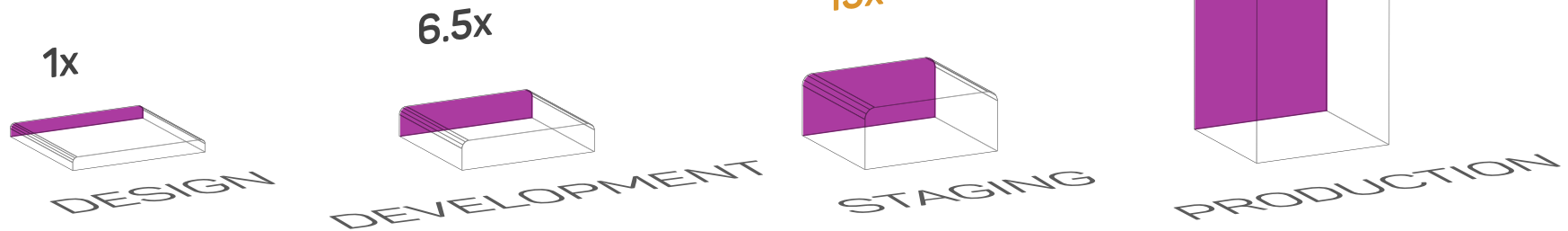
Testers continue to wait for new builds, and also wait for developers to declare code-complete so that regression testing can move ahead. A lot of time is spent idling or trying to avoid being idle. But things can be - and must be - much better! Productivity can increase significantly when testing is done further upstream.

On too many projects, and even with too many agile teams, testing is low-priority consideration because it's much easier to get developers together with product staff well upstream in the delivery pipeline. It's much easier to focus on building the product. A succession of development

sprints continues, but it's difficult to convince product and development to weave testing into each iteration. Early testing is often a struggle.

The result is that testing gets squeezed into a short window prior to customer release. As many teams will attest, this passive approach to testing creates quality problems and builds tension between developers and testers. In many companies, this is lamentably the way things are done. For managers that want to move faster, testing is seen as a bottleneck. It's important to realize that testing only becomes a bottleneck when testing isn't approached as a whole-team effort.

THE HIDDEN COSTS



Relative Sosts to Fix Software Defects (IBM Systems Sciences Insitute)

Beyond the cultural burdens that the traditional waterfall development model brings, you have the exponential costs of growing technical debt when fixing software defects in production. According to the IBM Systems Sciences Institute, the relative

cost of fixing defects in production or maintenance phase of the development life cycle is 100x more than the design stage when testers can be involved to drive quality assurance and testing efforts in subsequent phases of development.

THE BENEFITS OF SHIFT-LEFT TESTING

Shift-left does more than help your team find bugs earlier. It can also help a team collaborate better with all stakeholders, improve collective competency, and craft more realistic test cases.

Shift-left testing brings with it several cultural benefits since it places greater emphasis on these areas (from the well-known principles of the agile manifesto):

- Customer collaboration is more important than contract negotiation.
- Responding quickly to change is more important than strict adherence to a plan.
- Interactions among individuals are more important than processes and tools.
- Working software is more important than comprehensive documentation.



In summary, shift-left testing enables:

- More intensive focus on customer requirements leading to better product design and user experience.
- Early, progressive, continuous testing that reduces the number of defects.
- Increased efficiency and reduced technical debt, reducing costs.
- Improved software team culture, competency, morale, and employee retention.

Shifting left means getting everyone on the delivery team engaged in testing activities. Testing changes from a scramble right before release to something your team talks about and does every day. Let's look at some healthy changes you should be making to your testing processes when you adopt a continual, holistic testing approach.

SHIFT-LEFT TESTING RESHAPES PRODUCT DEVELOPMENT

Shift-left testing doesn't mean that no testing occurs in production, or that testing is completely shifted to only the design phase of software development. Shift-left injects testing into each sprint. Some testing should still occur at the very end, but it should be residual. It should be also be performed relatively faster since most of the problems should have already been found and mitigated.

This shift won't only result in some early design changes, but the testers will learn first-hand about the ultimate standard for testing the release. For example, testers

may realize that it's much more efficient to work closely with component and system developers as they learn about the product specs. They can ask probing questions, begin forming testing approaches and formulate testing scenarios. Other testers might meet with the API developers and work to create test stubs for new services.

As testers are actively participating more in these earlier phases, they are effectively "shifting left" in the waterfall sequence of events in the software delivery pipeline. In practice, testers can progressively test new features as they are made available.

DESIGN PHASE: TESTING FEATURE IDEAS

As the product team and delivery teams learn how their customers use the product and what value they are still seeking in it, they get new ideas for features. In Waterfall, the product design team would wait until they had a critical mass of new features, then start a long process of designing them. Testers might not have even been aware that new features were underway. In modern software development, we can test new feature ideas right away, by asking questions like:

- What's the purpose of the feature? What problems does it solve for our customers? Our business?
- How will we know this feature is successfully meeting customer needs, once it is in production?
- What's the smallest slice of this feature we can build and use as a "learning release" to make sure it has value?
- What's the worst thing that could happen when people use this feature? What's the best?

USING TESTS CONTINUALLY TO GUIDE DEVELOPMENT

Here's a typical "shift left" scenario: Your team has decided to build a feature. The product owner has written an epic for it and the team has sliced it into small, testable stories. You're having a specification workshop or a discovery session with a small group to discuss the stories before the planning session with the whole delivery team.

Refining the Specification: An Example

Free Delivery

Free delivery is offered to VIP customers once they purchase a certain number of books. Free delivery is not offered to regular customers or VIP customers buying anything other than books.

Customer Type	Cart Contents	Delivery
VIP	5 books	Free, Standard
VIP	4 books	Standard
Regular	10 books	Standard
VIP	5 dishwashers	Standard
VIP	5 books, 1 dishwasher	Standard

Specification by Example: How successful teams deliver the right software, Gojko Adzic, pg. 116

Kicking this type of discussion off by asking “How will we test this?” leads to a productive discussion. The discussion can help you understand the feature from the standpoint of individual stories and this will lead to how to implement tests for those stories at all levels including unit, service level/API, UI, or other levels as appropriate. The team will design testable code, which leads to more reliable code that is easier to understand and maintain.

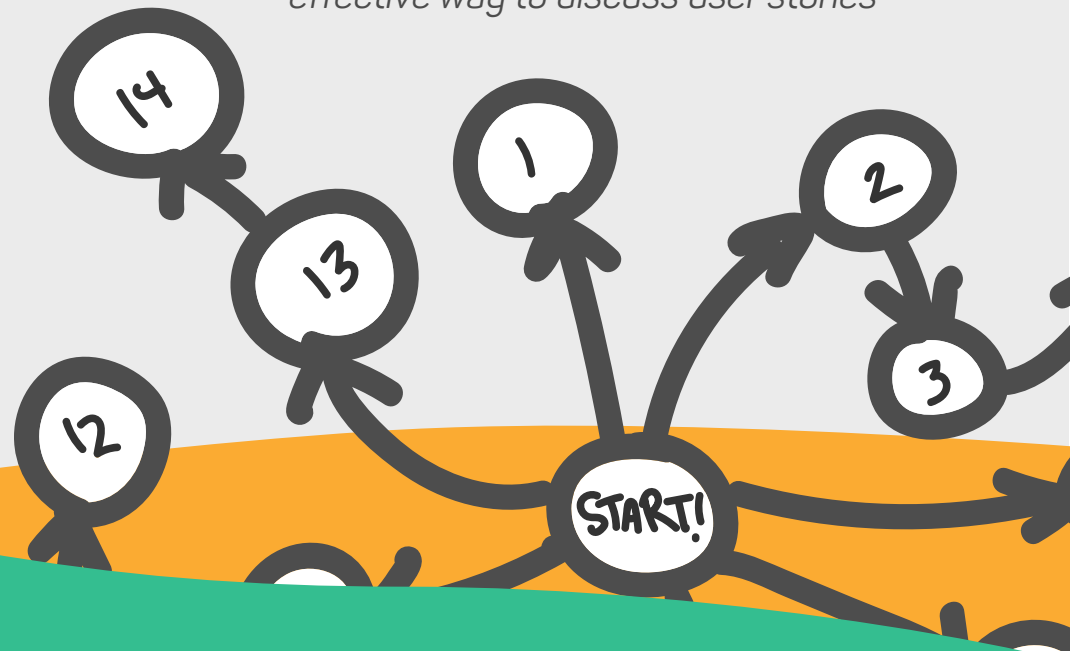
DEFINING TEST PLANS

Instead of formal test plans, capture a few examples of desired and undesired behavior for each story as they are broken down. The team can turn those into executable tests that guide development. Once the feature is in production, these tests become living documentation of how it works, and automated regression tests can make sure future changes don't break it. As each feature is built, more test cases will come to mind to automate or explore manually, which is one of the ways to improve your team's chances to deliver exactly what your customers want.

For most business domains, the days of heavyweight test plans are over (it's

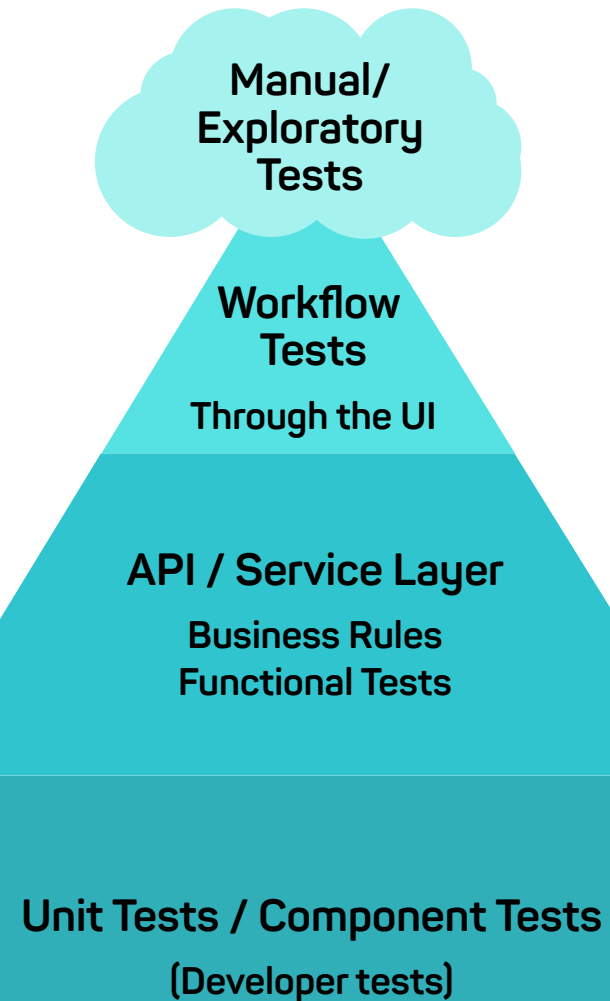
questionable how many people ever read them anyway!). Many teams find mind maps are a better solution for planning and tracking testing. A simple one-page test plan can also work well, as well as a test matrix written on a whiteboard. Your team should be given the freedom to experiment with different formats to see what works best for them.

Mind mapping on a whiteboard can be an effective way to discuss user stories



TEST EARLY AND OFTEN, AUTOMATE EARLY AND OFTEN

Models such as the test automation pyramid help guide discussions around how to automate various types of tests. Such frameworks provide a vehicle for discussing how to automate testing for a particular feature at different levels, which helps to share the testing burden across both developers and testers. Developers may discover that it would be easier for them to test-drive their code with API-level tests, or, they may find that they can test everything at the unit level and no higher-level tests are needed.



In agile development, features are built incrementally which means you can automate as you go. Even if a capability isn't fully built out, you can write the most basic test for it and build onto both the production and the test code for this capability in pieces. By adding and expanding automated tests with every incremental delivery, you avoid a scramble to automate tests last minute, and even better, avoid putting test automation off until a future iteration, which would add technical debt.

If some or all of your product isn't yet supported by automated tests, that's another challenge for the whole team to tackle. Nobody has time and money to automate everything, especially in the short term.

It would be beneficial for the team to get a representative cross-functional group together and draw the high level system architecture on a whiteboard (real or virtual), and identify the riskiest areas to make intelligent decisions about what to automate next. A framework for risk-based discussions is Ashley Hunsberger's Test Suite Canvas, pictured on the next page.

When testers, developers and other team members collaborate to automate tests as coding proceeds, there's less waiting around to get questions answered, more expertise to write maintainable tests, and less rework. Quite a contrast to the handoffs in waterfall processes.

A framework for risk-based discussions, Ashley Hunsberger's Test Suite Canvas can help your team make intelligent decisions about what to automate next.

TEST SUITE CANVAS:

Why	Dependencies	Constraints	Pipelining/Execution	Data
<p>What business question am I trying to answer with the suite?</p> <p>What risk does this suite mitigate?</p>	<p>What systems or tools much be functional for this suite to run?</p>	<p>What has prevented us from implementing this suite in an ideal way?</p> <p>What are known workarounds?</p>	<p>Is the suite part of a pipeline?</p> <p>When is it triggered?</p> <p>How often does it run?</p>	<p>So we mock, query, inject?</p> <p>How is test data setup/managed?</p>
<p>Engagement and failure response</p> <p>Who created the suite? Who contributes to it now? Who is not involved but should be? In the event of a test failure, who addresses failures and how?</p>	<p>Maintainability</p> <p>What is the code review process? What documentation exists?</p>	<p>Effectiveness</p> <p>How do we know the suite is effective? What is it finding? What is it preventing?</p>		

IT'S NOT JUST ABOUT TESTERS

Though we've been explaining how testers need to get involved earlier by shifting left, the same goes for developers who need to shift right and get more involved with testing. Developers need to write tests as part of writing new features. Testers can work together with developers to walk through how to write effective tests, and once written, tests can be managed and maintained like code where developers make updates to the tests along with core product code changes.

BDD (behavior-driven development) is another strong collaborative process that can include even non-technical stakeholders in


the design phase of a new feature. BDD gets the team, including business stakeholders, to discuss the desired behavior of a unit of software based on requirements driven by the business goals, and more importantly, outlines who the ideal user in mind is. As a result, benefits that this unit of software should achieve for the ideal user is top of mind for everyone. In other words, BDD is an "outside-in" activity as the business analysts collaborate with developers, testers, and product owners to define the behavior of the feature from the perspective of the business value it delivers.

TDD (test-driven development) is about automating tests and running them early and often, which often involves automating more than just the tests. First, the infrastructure for this needs to be in place. This infrastructure needs to automate the deployment of new builds with continuous integration and delivery pipeline management, source control, and issue tracking platforms that work well together. Equally importantly, find a testing platform that is going to integrate into all of the pieces of that pipeline.

Consider implementing a testing-as-a-service platform such as mabl that can be easily triggered to kick off end-to-end UI tests automatically on new deployments,

integrate into Jira and Slack to create issues when defects are detected, and runs in the cloud with on-demand resources to reduce the infrastructure management burden on your team.

Of course, you can't expect a well-oiled machine overnight. Start slow; implement the shift-left methodology to one small team first and have regular meetings to discuss how success is measured. Discuss how to overcome challenges the team is facing with the new process implementation. Document the process once the kinks are ironed out, then roll it out to the rest of the teams.



OTHERS HAVE MADE THE SHIFT - AND YOU CAN TOO

Getting involved with testing at all points in the continuous development cycle can be daunting, but there are plenty of success stories from the testing community that show that it's possible no matter what your respective role is.

Andrew Morton, Isabel Evans, and Ken Talbot wazill reveal what first steps they took and how they snowballed into a movement that benefited everyone around them.

ANDREW MORTON

Now the DevBoss of Ministry of Testing and [@TestingChef](#) on Twitter, Andrew used a fairly radical approach to “shift left”. He decided the best way to show developers how to test was to be a developer who tested. If you have some coding experience or are interested in learning, combining your testing expertise with hands-on development work is a great option.

*“I knew that to be as effective as possible, I needed to **be around when decisions about a project were being made, from initial kick off onwards.***

*To this end, I used to **check my colleagues' calendars each day, and if there was a meeting scheduled I thought I should be in, would ask to be invited.** This was the*



first step in shifting left, as I was now able to be involved at the start of a project, and when features were released to testing.

*However, there was still a gap that meant I wasn't testing throughout the entire development cycle - and that was the part where code is being written. In theory, as a tester, this is not my realm to work in and developers should be doing things required to check their own work. However, over the years I have noticed that **most developers are not taught testing (rather, they pick it up 'on the job' and learn from code written by others in the same situation),** and even if they are, they are not necessarily great at applying it.*

The question became how do we fix that? The company I was in tried moving testers

out of the dedicated testing role and into a test coaching role (i.e. we wouldn't do any testing ourselves, but help the development teams do testing instead). Whilst this kind of worked, it wasn't very satisfying to me, because I had the feeling (probably erroneously, I admit) that I wasn't getting across to developers because I wasn't one.

So I changed that. **Deciding that the best way I could show developers how they could test and develop, was to become a developer who tested and developed.** I had quite a bit of programming experience from doing E2E automation, and had just come off a project backfilling unit tests to replace some integration tests that could be run at the lower level. I was now in the position of being able to be part of design, programming, and testing

for features that I was assigned. That approach has stayed with me for the last couple of years as I have continued my developer journey."

Andrew's desire to "shift left" led him into a new role where he could model good testing practices to other developers, which was an effective way to help them learn testing skills. Testers pairing with developers to automate tests, do exploratory testing, or write production code can help transfer skills among team members so that they can prevent bugs and please customers.

The first step in shifting left: be around when decisions about a project are being made, from initial kick off onwards.

ISABEL EVANS

A well-known software and quality consultant and international speaker with more than 30 years experience in IT, Isabel has seen testing change... and *not* change. She remembers that the actual correct approach to testing in waterfall was originally to do testing at each phase, rather than only at the very end of all the phases, right before release. Here's her story about continuous testing:

"I went to a training day run by Bill Hetzel and he was talking about the importance of 'Test then code'. I went back to the place I was working and started to implement it.

As a system tester - the only one in the company - I wrote a report to the



managing director to say 'we could do things differently and it would save time, money and hassle' and he asked me to implement it.

*I implemented it first with discussions, through testing requirements, through testing design stages, to unit tests designed before coding, and then testing running through the stages from unit to integration to system to acceptance testing. **We had a 'stop and think and test/review' at every stage in our projects.***

My job was to train them in testing, and in review/inspection methods, and in how to tailor those to risk and project size. I would coach, help etc. Implementing this took time, and building of trust, and demonstrating that I could test, and find useful improvements, and help them ditto.

I saw my role as serving everyone and enabling them to succeed.

We (we were a software house) also went out and taught customers how to test and how to review so they could join in at all stages, and report back on anything live.

And we also involved a technical person in each sales proposal, and proposal documents were also reviewed and tested.

So in the end, testing started at presales and went on into post delivery customer care.

I am still proud of what I achieved there. I'm still in touch with some of those people. At his 70th birthday party, Managing Director made a speech about the effect I'd had on the company - it was his birthday, but he chose to speak about the difference end-to-end testing made.

Nowadays, I am taking the shift even more seriously - if you are focused on User Experience (UX), it is even more important that you start at idea stage with testing and never stop. Keep retesting your ideas, your preconceptions of who the users are and what they are doing, and what the products provide for customers."

When Isabel says "end to end testing" there, she doesn't just mean some automated test that encompasses a full user journey through all layers of an application. She means testing from the idea end of a product to the customers using the resulting features in production - and as well, taking what's learned from customers and feeding that back into the next new product ideas.

KEN TALBOT

An English and Film graduate who started out on a path to be a writer and journalist, Ken's interest in exploring tech and software led him to working on a small test team where the development followed a waterfall approach. When the company transitioned to agile development, Ken was able to learn much by communicating directly with developers, system administrators and database engineers on a daily basis. He was able to shift away from the waterfall testing-at-the-end approach by applying his natural curiosity - and some courage.

“I have found - at two companies so far - that an inquisitive nature, coupled with as much confidence you can muster, enables shifting left/right. I held a mob testing session with some



new testers and graduate devs. The devs turned into inquisitive testers after a few turns and the testers were asking questions about programmatic design. This is something they confessed would never have happened in their project team.

After a recent visit to a local tech expo, our company's engineers were evangelising about the DevOps ecosystem and how 'everything is a dev problem now' as cloud architecture and infrastructure as code managed by the team no longer requires the traditional Dev-QA process. This understandably impacted the confidence of some of the testers, particularly the non-technical demographic.

I felt the need to assure teams that not only was the 'everything is a dev problem'

*somewhat misinterpreted, but that if we used that logic then test is a dev concern and dev is a test concern. **Identification of risk and implementation of test structures - automated or otherwise - still requires areas of the team to specialise in these factors. Seeding of that knowledge must begin somewhere and this is why testers must always be involved in all stages of the development cycle, especially in the architectural planning of the shiny new cloud-based world we live in.***

Since then, I believe all of our testers have continued to not only take an active role in development, but also start to broaden their knowledge of newer technologies to better assess the changing tech landscape."

Testers can make their own opportunities to start collaborating with developers, operations experts, designers and others to move into testing ideas on the "left", and learning from production use on the "right" - along with everything in between. Testing is an integral part of product development, not a "phase" to be done at one particular time in the cycle. Don't be afraid to take your first steps to a more holistic testing approach.

An inquisitive nature, coupled with as much confidence you can muster, enables shifting left and right.

WHAT MANAGERS CAN DO


From a management perspective, you may face multifaceted challenges that can easily halt your progress or kill your motivation. But every challenge that you may face will most likely be rooted in resistance to change. You can address this with transparency: proactively by setting the right expectations, and reactively by instating a channel that gives the entire team a window into the results that shifting left has on product quality. Here's a few things to consider.

MAKE TESTING A WHOLE-TEAM RESPONSIBILITY

If you want to be delivering a quality product at a consistent basis, then everyone needs to have a stake in the quality of the product. That includes developers writing unit and integration tests, quality engineers writing API tests and UI tests for critical flows, and then also collaborating with the product owner and the developer to automate strategically to make sure we were covering the highest priority cases.

SET REALISTIC EXPECTATIONS

When your team begins to write more unit, integration, API, and end-to-end tests, you're



going to need buy-in from both product and engineering to deliver less features in a sprint in order to ensure that the tests your team has committed to writing in that sprint are actually created.

KEEP THE ROI TOP OF MIND

After reducing how many stories or features should be delivered per sprint, you need to remind your team of time they're saving by having a test-first mindset. Help your team quantify the ROI of finding bugs earlier on and addressing them as quickly as possible. Not waiting until bugs are found during regression or by a customer ultimately saves them time which allows them to deliver faster.

METRICS, METRICS, METRICS

The team needs to constantly see how the shift-left movement is benefitting the overall quality of products. You can measure the number of bugs found, but another way that encourages a test-first mindset is implementing a quality index scale. Value is assigned to a bug based on where it was found (ex: the further along the bug is, the higher the percentage). As a result, time-savings can be more easily quantified and encourages the team to keep improving on time to resolution opposed to finding an arbitrary number of bugs.

CONCLUSION

Shift-left testing is a movement that the entire software team is involved in. With transparency, buy-in from all work units that need to be involved, the right expectations, and measurements in place, shift-left testing can bring great success to your team and business.

To summarize, here are some of the adjustments that each role needs to make to support the shift-left testing movement.

TESTERS: Inject yourselves into the design phase. Don't wait for a build to become code complete to test it. Endeavor to break down user stories into small testable chunks to ensure tests are effective and meaningful.

DEVELOPERS: Take part in discussions around test planning and strategy. Your skills can be valuable for unburdening the testing team and reduce technical debt for yourself later on, too. View your automated test suites as a core part of the code base and maintain it.

PRODUCT MANAGERS: Allocate the proper resources for your teams to collaborate and facilitate feedback. Adjust your expectations and set goals that harmonize with the overall

shift-left testing strategy. This may include expecting less feature-work per sprint and setting quality goals for the entire team.

INFRASTRUCTURE AND DEPLOYMENT

TEAMS: Testing needs to happen early and often, which means your deployment pipeline needs to be available early and have a high capacity so that tests suites can run in a timely manner. Consider testing-as-a-service platforms like mabl for on-demand testing resources so you can run all your test suites in parallel.

It's no longer a question of why, but a question of when. Why not start the discussion with your team today?

RESOURCES FOR CONTINUED LEARNING

“Shift Left - Why I Don’t Like the Term”, Janet Gregory, <https://janetgregory.ca/shift-left-why-i-dont-like-the-term/>

“Continuous testing in DevOps”, Dan Ashby, <https://danashby.co.uk/2016/10/19/continuous-testing-in-devops/>

Discover to Deliver: Agile Product Planning and Analysis, Ellen Gottesdiener and Mary Gorman, <https://www.discovertodeliver.com>

“The Learning Release”, Ardita Karaj, https://medium.com/@Ardita_K/the-learning-release-70374d2450b3

“Are we shifting left or in a continuous loop?” Augusto Evangelisti, <https://mysoftwarequality.wordpress.com/2018/04/28/are-we-shifting-left-or-in-a-continuous-loop/>

“Key skills modern testers need”, Interview with Janet Gregory on StickyMinds <https://www.stickyminds.com/interview/key-skills-modern-testers-need-interview-janet-gregory>

The Three Amigos Strategy of Developing User Stories, <https://www.agileconnection.com/article/three-amigos-strategy-developing-user-stories>

Discovery: Explore Behavior Using Examples, Seb Rose and Gáspár Nagy, <http://bddbooks.com/>

“The One Page Test Plan”, Claire Reckless, <https://www.ministryoftesting.com/dojo/lessons/the-one-page-test-plan>

“Monitoring and Observability”, Cindy Sridharan, 2017, <https://medium.com/@copyconstruct/monitoring-and-observability-8417d1952e1c>



The 1st DevTestOps platform

helping software teams shift testing left and right

[Create a free account](#)

www.mabl.com