WHAT DOES QA STAND FOR IN SOFTWARE?

QA, short for “quality assurance”, is the process of ensuring a high level of quality in software products. You probably already know this. But QA doesn’t just mean testing. There’s much more to it than that. Let’s take a look at what exactly “QA” encompasses.

**What does QA Stand for?**

**QA is a mindset.**

QA in software is more than a long list of testing activities. It is a mindset your entire team should embody. QA includes *everything* that has an influence on creating high quality software products. That includes processes, procedures, tools, people, and standards. QA means always keeping a pulse on weaknesses and identifying opportunities for improvement in all of these areas.

If your team is constantly pushing out new features, and never revisiting them to improve quality (even if said features don’t have any bugs), you have a quality problem.

**Testing doesn’t create value, it defends it.**

Testing alone cannot guarantee a high quality product. Think of an app that is free of bugs (rare, if not impossible), runs quickly, has a slick UI, but it does not help it’s users achieve what they need to do. Would you still consider it a high-quality application?

In QA, testing doesn’t necessarily “add” value to a product, it ensures the value is there. As a tester, your job in QA is to ensure wide test coverage and to always speak up when you identify an area of weakness in the product or process.

**QA encompasses the entire development process.**

Quality needs to be the focus from the very start with requirements gathering, and all the way through to maintenance. This involves a wide range of testing techniques, documentation, and processes. Once processes have been defined, QA is responsible for identifying any faults in the process and correcting those weaknesses to ensure continual improvement. A good QA professional keeps an eye on the bigger picture. They help your team hone in on quality issues from communication to code.

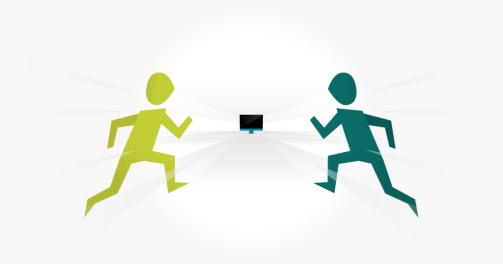
The entire development process is prone to weaknesses and improvements in quality. It is QA’s role to keep this at the front of everyone’s mind.

**Conclusion**

As you can see, QA is not just the act of testing. It is the belief that every part of the [software development life-cycle](https://blog.testlodge.com/software-development-life-cycle/) has the opportunity to influence quality. A QA mindset can help built quality into the product from the start. If you have a quality problem, don’t just hire a quality team. That’s more of a band-aid approach. Instead, solve the problem at a deeper level. Hire quality-focused people, and always take a step back to make sure you’re heading in the right direction – towards quality.

QA Can Speed Up Software Development

It’s not uncommon for QA to be thought of as a roadblock, existing only to slow down the [software development lifecycle](https://blog.testlodge.com/software-development-life-cycle/), raise bugs, and tell developers their code is broken.  Well, those are indeed some of a tester’s responsibilities.  And it’s true; QA can slow down development, but it’s important to consider how in the long-term, this can actually speed up the overall software development process.



In other words, you can actually speed up development by slowing down.  Seems odd, right?  Here’s how…

**QA testers find bugs before the customer.**

If QA is doing their job, they should be finding bugs before the customer does.  By the time bugs are introduced into production, your developers are already building the next feature.  Their focus is on something new and different; not what’s already been shipped.  When a bug is introduced in production, this doesn’t just look bad to your customers, it requires your developers to stop whatever they’re working on, shift gears, and refocus on something they’ve been away from for days, weeks or even months.  This is a huge hit to productivity.  With proper testing, you can avoid those surprises and keep things moving in a forward direction.

**QA testers can identify usability issues early on.**

Similarly, QA testers are excellent feedback providers.  QA can identify issues with usability before the user is even exposed to them.  This saves the time of having to re-design and re-code features after they’ve already been shipped.  When [QA is integrated with the development team](https://blog.testlodge.com/qa-and-development-working-in-parallel/), quality becomes an inherent trait of the entire team.

**Dedicated QA ensures developers are focusing on development.**

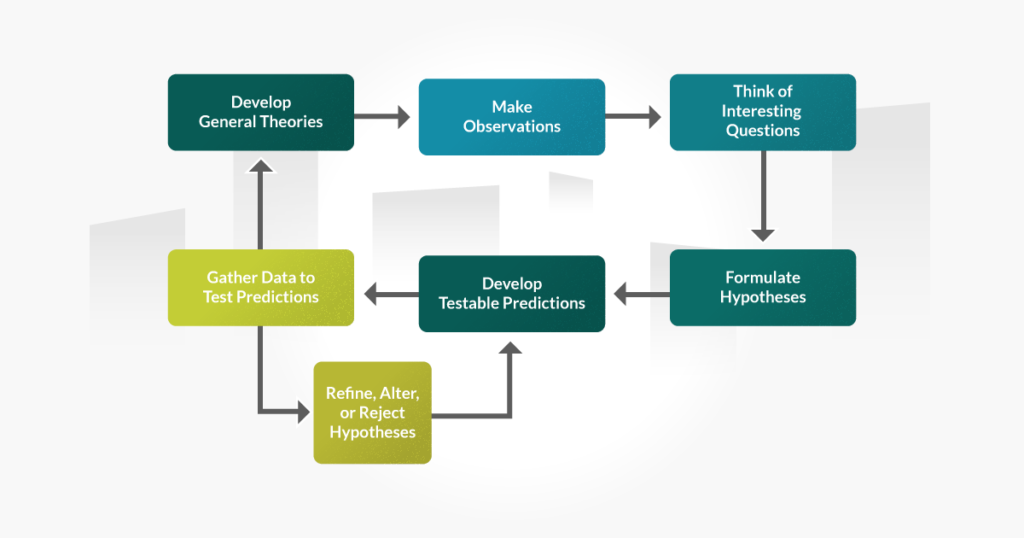
Developers are employed to write code.  Let them write code and stay focused on that.  If you practice test-driven-development, keep doing it, but let QA handle the real heavy parts of testing before release.  Integrate testing into the development process and you’ll find that developers are more efficient**.**

**In Conclusion…**

In the agile world, we’re focused on working fast.  After all, that’s part of being agile.  But it’s important to let QA apply the breaks once in a while.  This doesn’t mean your QA team should always be testing everything.  Work fast when working fast makes sense.  Identify areas of risk, and let QA provide feedback, investigate, and test.  The sooner you discover an issue or error, the sooner you can get back on track to building awesome software.  By closing this loop and injecting quality into the process, you’ll end up with a higher quality product and your team won’t be running around distinguishing self-inflicted fires

Applying The Scientific Method to Software Testing

Anyone with children (or in my case, nieces) will know that they spend their days trying to make sense of new experiences in the world around them. They explore, they discover and without even realising it, they employ The Scientific Method.



Almost a thousand years before the birth of modern computing, the polymath Ibn al-Haytham, advocated the importance of asking questions and then testing them. Galileo built on that and detailed his experimental methods in 1638 with the publication of “Two New Sciences”. He believed that knowledge could be acquired through reason, then reinforced through observation and experimentation. The Scientific Method has been honed for this very purpose and here, I will cover the steps involved. These steps are not set in stone and would be best considered as a set of general principles to work by; use them as another tool, not a straightjacket.

**6 Steps of the Scientific Method**

**Make Observations**

“You can observe a lot by just watching.”

*Yogi Berra*

Testers work in a similar manner, ferreting out the truth from the myths and outright lies. I like to start right at the top of that diagram. I just dive right in, begin testing and making my own observations on the fly. This can be anything and everything from overall functionality to specific functions or aspects, tools and widgets, to page layout, look and feel, ease of use and readability.

**Interesting Questions**

All the time you observe, think of interesting questions. Are page load times acceptable? Does JavaScript slow things down unacceptably? Is the page or site navigable? Does that web form function as expected? Is this page too graphics heavy? I once tested a website with a very high-resolution BMP image in the background that doubled the size of every page it was used on and yes, it was used on every… single… page.

**Formulate Hypotheses**

Keep asking questions and you will start to get an idea of what is happening with the system under test so you then have hypotheses to work with. Depending on how much access to the underlying code you have, you may have a very good idea of the internal goings-on. Perhaps you just take a black box approach to testing.

**Testable Predictions and Gathering Test Data**

Whichever way you have observed system behaviour, you will then know that you can make an intelligent, informed and testable prediction (as opposed to just a wild guess) about how the system behaves, and will behave, given a specific set of inputs.

Gathering data is probably the part of this method testers are most familiar with and spend the most time doing. You test the testable prediction, push the envelope of your hypothesis and do everything you can to either prove or disprove yourself. I’m not sure which is the most satisfying event as a tester – that “Eureka!” moment when something turns out exactly as you predicted during a test, or the instance something unexpected happens and you wonder out loud “Hmm, that’s funny…”?

**Be ready to adapt your Hypothesis**

The reality of testing falls somewhere between the two stools mentioned above. At those all too frequent points, you get up, rub your bruised elbows and have a serious think about what’s happened while you refine, alter, expand or reject your hypothesis. It may just need a tweak or two, or you might have to go right back to the drawing board (or Kanban board).

Try not to lose heart if you find yourself back at square one, because it’s not really square one any more. Every test you do will in some small way add to the sum of your system knowledge. As you work and document the testing processes and results, you are creating your own knowledge base built on all of your observed results.

**Develop General Theories**

The payback at the end of all your extensive testing work is that you should now be able to develop general theories about the system under test. Once you get this far, testing should, in theory, get easier because by this point, you’re not a mere tester, but have become a product expert.

Now you have a general working theory, you can make even better observations, with greater insight and your predictions are likely to be more refined and accurate. Exceptions to the general rules that you’ve hammered out will be much easier to spot. You’ll go back to making observations again, but this time with more testing experience, confidence, knowledge and wisdom.

**Conclusion**

Consciously or unconsciously, we all adopt these processes to some degree in our daily lives, but The Scientific Method only gets serious mileage and attention in fields where the pursuit of knowledge is the primary goal. Kids are like sponges, absorbing as much information as they possibly can every second of every day – Scientists and Testers can learn a lot from that.

3 Common QA Challenges

In software testing, there are many QA challenges. With so many different types of software applications out there, there are numerous issues to be overcome by every QA (Quality Assurance) team. Let’s talk about a few of the common situations faced by just about every software tester, and take a look at some solutions that can help improve the quality of your product.



**Challenge #1: Keeping up with Requirements**

Requirements can change quite rapidly for a lot of software development teams. Testers must be prepared to adapt to those changes and know what to expect when a build becomes ready for testing. Keeping up to speed requires having a plan for updating test cases and testing the right scenarios.

Should you update the test cases immediately after receiving the updated requirements? Which test cases will need to be updated? Do you need to update for regression tests? What is the process for notifying the QA team of requirement changes?  – All these questions can be answered by defining a process for updating documentation.

Ideally, the development team should have a plan in place for evaluating the updates and taking action. Sometimes requirements change at the 11th hour, and testing must be executed quickly to meet a deliverable. In these cases, QA must be flexible yet steadfast in the test documentation. Assigning scheduled time for regular test updates is the key to staying on top in this area. This scheduled time could be based on Sprint work within an Agile environment where each user story is considered for a test update. Another method could be to update tests for each release.

The frequency and cadence of the test updates could vary depending on the product and the development life cycle. Once a plan is in place for scheduled test case updates and maintenance, it is essential to stick with it because obsolete test cases become a significant hindrance. They result in slower test execution, cause bugs to be missed entirely and can ultimately be the reason software fails to meet the latest expectations.

**Challenge #2: Balancing Sprint Work with Releases**

Testing software in an Agile development framework brings several challenges to the QA side of things and the [role of QA in an agile team](https://blog.testlodge.com/the-role-of-qa-in-agile/) may differ when compared with other project management methodologies. One of the benefits of testing in Agile is the chance to look at changes early and often, which means the software should be ready for the QA team to test more quickly after the requirements are created or updated. This quick turnaround gives the QA team a chance to catch issues early and provide the team with feedback straight away. The challenge can be keeping up with sprint work while accommodating regression testing for a release.

The work done within a sprint is typically a different set of tasks than those required to regression test and certify a release. QA must, at times, devote resources to releasing testing and sacrifice time from sprint work. Sprint work is the development and testing that occurs (usually in a development environment) within a given time frame. In agile a sprint can last anywhere from 1 to 3 weeks. If some of the sprint work is being done for a release down the road and QA is testing for an immediate release, then the attention given to the sprint may lessen.

The key finding a balance lies in process and planning. A proper backlog of testing tasks should be maintained, so as not to lose track of time missed while testing a particular change or set of changes. The QA team should communicate the workload that is required and point out any areas where there are gaps in testing. Likewise, the software team should be ready to adjust schedules and resources to account for these issues.

Often, QA will need to put in extra hours to accommodate regular sprint work AND a release. In the end, proper scheduling and communication should alleviate the need for testing overtime or leaving gaps in testing.

**Challenge #3: Deciding Where to Test**

Deciding where to test might not seem like a common challenge, but think about the previous issue we addressed – balancing release testing and sprint (development) testing. Release testing is generally performed in an environment separate from sprint work. So, the “where” in that case, is the build and the environment.

Often, there are multiple testing environments such as development, QA, staging, and production. The production environment is generally reserved for the production users, but it must be available for testing as well. The staging environment is typically setup to be the most similar to production, so testing here is reserved for making that final verification before going live. The dev and QA environments are setup differently but are both primarily used for testing.

The “where” includes the environment, the build (the software built with particular changes) and the client (such as the device, the OS of the device, the version of the browser, etc.) There can be more variables when you have multiple “environments” that an app or website hits to retrieve data to display to the end user. So, a mobile app, for example, could hit a development server for one functional area and a production server for another. The setup of the environment is the “where” in many cases.

The keys to this challenge are an evaluation of priorities, assessment of risk and an understanding of the environments. Ideally, QA can follow the changes through the development cycle and test them at every point along the way. In the real world, there are days where a decision must be made to test the upcoming release candidate build instead of the build in development. A plan should be formulated with the development team to understand the risks and priorities. The environment setup is crucial when you are testing a client. If the environment isn’t setup for the function under test, you will end up missing bugs, reporting invalid bugs or be unable to test in that environment.

**Conclusion**

There’s a common theme when it comes to finding solutions for these challenges, and it’s process planning and communication. Make plans for handling documentation and make sure they are organized and kept up to date. Communicate with the developers, the project team and the release engineers. These are the kind of challenges that QA faces on a regular basis. Get used to solving these, then get ready for the next challenge.

Software Testers – Take Control of Your Time!

Testing takes time, a lot of time. And it’s never enough. Everything can’t be tested all the time or all at once, but what we can do is use our time and energy wisely. Even small tweaks made to test processes can save hours and days in the long run. This post is about highlighting some of the major QA time killers and offering practical ways to avoid, or overcome them. Some time wasters are super-obvious and others are not so evident – but trust me they are killing your efficiency. Here, we explain why.

**Organize Regression Tests**

[***Regression testing***](https://blog.testlodge.com/what-is-regression-testing/)is one of the most time consuming tests we run, so let’s start here. The goal in regression testing is to make sure that recent changes (no matter how big or small) have not unexpectedly impacted the software’s functionality. Some form of regression testing needs to be executed prior to each release to production but exactly how much, depends on the changes made to the release candidate.

Regression testing is a great candidate for making test efficiency improvements because of the frequency at which it is executed. This is also why regression tests are the first tests to be considered as automation candidates. So, making these manual tests quicker and easier to run can save a bunch of time and energy in the long run. Here’s how to shave time off of regression tests.

**1. Order test cases by feature and take a look at the flow of your tests**

If you are testing part of one feature, then jumping to another feature before going back to the first, you can avoid extra taps or clicks by grouping tests together more effectively. Be certain theses moves make sense, though. If the software is designed for a user to toggle between features, then it might make sense to test that same path as well, but during regression testing, this might not be necessary because those scenarios could be tested as part of other test cases.

**2. Order tests by build type and/or environment**

Some testing may require a debug build or a separate environment in order to use testing tools for those features not exposed to the end-user. If the goal is to approve a build for release, you will want to reserve the debug build for only the tests that require it. So, group your tests by those that require a debug build and those that don’t. This will eliminate the need to jump between the two builds or environments. Not doing so can be especially time consuming if you need to do things like clear data or uninstall and reinstall.

**3. Order by hardware or browser**

You probably get the idea by now. It might not seem like it takes long to switch between a phone and a tablet while testing a mobile app, but the time it takes to unlock the device navigate to the screen you are testing etc., will add up. Shaving minutes or even seconds off tests that are repeated often will multiply into substantial time savings over the course of a year.

**Improve Requirement Clarity**

The importance of concise, understandable, [quality requirements](https://blog.testlodge.com/writing-better-requirements/) is key to software development efficiency. The quality of requirement communication will determine the level of harmonious understanding between tester and developer (and ultimately product.) When requirements are vague and left open to interpretation it can lead to time-wasting questions, development re-work and then additional testing. Requirement questions and clarifications at the time of testing are big time wasters for QA. Instead of verifying the functionality and moving on to the next item, there is a back and forth between product-QA-development that eventually depletes everyone’s time and energy.

So, how can we avoid this situation? Of course, getting the work done right is most critical and if in doubt, questions must always be asked. But, to avoid these scenarios late in the day, QA can be proactive by anticipating these questions during the review of requirements. If a requirement appears vague or leaves out any necessary details, bring it up before development begins. Make sure development, product and QA are on the same page. The time saved by clarifying requirements up-front could end up being the difference between meeting and missing a deadline. Or even worse – rushing a bad product out the door!

**Rabbit Holes**

We’ve all been there. You encounter a bug and can’t reproduce it or someone else reports seeing a bug and you spend all day chasing it down. Sometimes, procedures like this are absolutely necessary. Perhaps there is a crash that you know will affect a lot of users. In those cases it’s perfectly understandable to drop everything and figure it out because you know this is going to be a gating issue. (You won’t release until it’s fixed.)

Other rabbit holes will be less necessary to go down. Before spending too much time tracking down [steps to reproduce a bug](https://blog.testlodge.com/the-importance-of-steps-to-reproduce-a-bug/) or figuring out the root cause of an issue, take a moment to prioritize. How widespread is the issue? How critical is the issue? Is it only an issue on OS versions that are used by a low percentage of users? Reflect on questions such as these before digging too deep (spending too much time on the issue.) Spending too much time on a lower level enquiry may prevent proper time being spent finding a bug that is more critical. In these cases, make a note to return. Finish other test cases then come back to it. You might just figure it out after you leave it for a while.

It’s not uncommon for QA to be sent down one of these rabbit holes after a high level executive, such as a CEO, encounters a bug (or the CEO’s cousin’s nephew’s 2nd aunt) and requests it be looked into. It may be worth looking at, but the same questions should be asked regardless of the rank of the person reporting the issue. In too many cases I’ve seen QA’s time wasted to satisfy a client or high-ranking executive. In these cases, the end result can be a lower quality product because of this ultimate time waste. At the end of the day, the level of quality will be be a direct result of where time has been spent.

**Distractions**

Distractions and interruptions are obvious time killers. I’m pretty sure everyone in about every industry could tell you that. Software testers are certainly not immune, and avoiding distractions while testing software may be one of the greatest challenges you face. If you aren’t careful, half of the day can be wasted away because of useless distractions. You have a browser with twenty tabs open, Slack open on a second monitor, a smart watch on your wrist, several mobile devices (including your personal device) sitting on your desk and of course all the interwebs you could possibly want in front of you on all of your connected gadgets.

Here are some tips for avoiding those distractions and completing tasks more efficiently.

**1. Try using just one monitor**

If only one monitor is required for a particular task, shut down the other monitor or just disconnect the external monitor and use the laptop. At times when we pause to think about the current task, we gaze over at the other monitor and spot an email or Slack message that just came in. You might not even read it because you know it’s unimportant, but suddenly ….. “where was I?”  It doesn’t take much to derail our train of thought. Along the same lines as using one monitor, you can also close your tabs or put the application you are currently working on in full screen mode. The idea is to narrow our vision to one task until it is completed. Then, you can respond to your text or look up the population of Uruguay.

**2. Disable notifications**

I have spent the majority of my career testing mobile apps on Android and iOS phones and tablets. It’s not unusual to have several devices of each platform on my desk. While testing an application, notifications become a huge distraction. Take the time to disable notifications from other apps when possible. Or, simply put the other devices in airplane mode until you need them. Just remember to turn notifications back on when you need to make sure the devices are in a real world state.

**3. Try the Pomodoro technique**

This time-management method uses work intervals that are separated by breaks. Normally 25 minutes is used as a work interval followed by a small break, but the time can be set to whatever suits you. The point is to increase efficiency by focusing on one task for a set amount of time. Set a timer to 25 or 30 minutes (you can download pomodoro timers from most app stores) and focus solely on that task for the set amount of time. It’s amazing what can be done when you dedicate yourself to blocking out distractions for even half an hour. Chances are, emails and Slack messages can wait until the period is up. Personally, I don’t use this all day everyday, but when I notice I am having a particularly difficult time focusing and getting tasks completed, this is the method I turn to. Distractions and interruptions be gone! For half an hour at least.

**Conclusion**

Time is our most valuable resource in all aspects of our life. I’m sure everyone reading this has heard or read a quote about the value of time. In the software testing world it’s no different. Recognize the unnecessary time killers in your work environment and act to eliminate them. By doing so, you’ll find yourself being more productive, organized, less distracted and delivering a higher quality product in less time.

Meeting Customer Needs

In the today’s world of SaaS products, it’s more important than ever to deliver a product and service that is meeting customer needs. Subscription based products and services require us to win our customers over and over again, monthly by month, or year by year. If we fail to deliver what our customers need, they can move on to a competitor. Keep in mind, it’s much easier to *retain* customers than it is to *acquire* new ones.

With this in mind, here are some tips for meeting customer needs, so that you can retain more customers.

**Meeting Customer Needs**

We must understand our customers’ needs before we can work to meet them. Once we understand our customers’ needs, delivering solutions that meet their needs takes a lot of effort, empathy, and creative thinking. In this blog we’re going to look at ways you can understand and meet customer needs.

**Identifying Customer Needs**

First, you need to get an understanding of customer needs. To do this, you must always have an ear to the customer. You need to work to get to know the customer.

Make time to meet with and talk to customers on a regular basis. Jump into the support queue once in a while and interact with your customers there. These conversations will uncover a ton of insights (feature requests, common pain-points). In addition to customer conversations, monitor social media. What are people saying about your product or organization on the web?

Surveys are great ways to ask questions in a systematic way. Embed surveys strategically in key areas of your app. For example, if your product is subscription based, prompt the user to answer a few questions before they cancel their subscription. A few simple questions can provide a lot of valuable data over time, which can help influence changes further down the road.

Conduct user testing to get a microscopic view of customer needs. User testing is a great way to validate feature ideas, and make sure you hit the mark for your customers.

Finally, attend conferences and meet-ups to get face-time with your customers. This is a great way to get one-on-one time with customers outside of a conference call or stuffy meeting room.

**Distribute The Learnings & Take Action**

As you gain an understanding of your customer needs, you need to disperse this information amongst your team. If you use a team chat tool, create a feedback room where you can drop in insights, or automatically pipe in survey responses. Run regular reports from your customer facing teams and use this qualitative and quantitative data to influence your roadmap.

**Setting Expectations With Customers**

Understanding customer needs is an on-going practice. It never ends, because customers needs constantly change. As you gain an understanding of customer needs, it’s important to set expectation with your customers. Setting expectations with your customers will help build trust, and your customer need to be able to trust you. If your customers don’t trust you, they’ll find someone else they can trust.

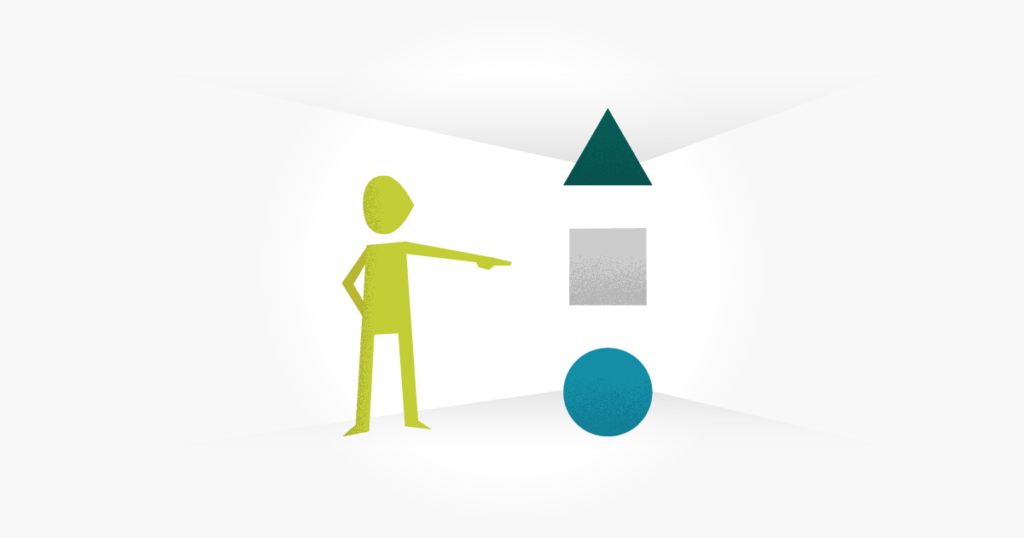
Communicate clearly with your customers no matter what the circumstance is. Through emails, in-app messages, blogs, support tickets. Follow up on feature requests, communicate quickly when things go wrong, and don’t give any empty promises. It’s impossible to meet every customers’ needs – just be open and honest.

**Conclusion**

Our customers are our business. The trick to meeting customer needs is figuring out how to make the biggest impact. Hitting the mark for the majority of your customers will have positive results. By understanding your customers needs, sharing the learnings & taking action, and setting expectations, you’ll maintain trust with your customers.

The Importance of the Expected Outcome

When I was a teenager, I received guitar lessons from a small shorts-wearing man called Matt…or Lee…, maybe Tom. It doesn’t matter what his name was; this was well over a decade ago and I’m pretty sure he doesn’t rely on referral business via software testing blogs. For the sake of this article though, let’s just say his name was Antonio and move on.



For about six months when I was about 15 years of age, each Tuesday at 10:20am, I would leave whatever class I had that morning (normally R.E. or Geography – an added perk) and walk across the school grounds to what can only be described as a large cupboard. In that cupboard sat Antonio, shorts and all, with two chairs, two guitars and two amplifiers (I was supposed to bring my own guitar to the sessions but regularly forgot, so he just brought a spare with him every week).

The half-hour lesson would then consist of me naming whatever song I was into at the time (probably ‘United States of Whatever’ by Liam Lynch, the biggest punk dropout song of them all in 2002), and then Antonio would try to teach me a pretty watered down version.

When it was my turn to have a go at playing the song, I would inevitably make mistakes (I am only human after all), and when I made a mistake, Antonio would tell me that I had played the wrong thing.

Where he really earned his ten quid lesson fee is that not only did he tell me that I’d played the wrong thing, he would also continue on to tell me what he was expecting me to play!

Mind blown, right?

Well, no, probably not. It would have been bizarre, awkward and down right unprofessional to tell me what I did wrong, but not tell me the expected result.

But man on the internet, what on Earth has this got to do with software testing?

Good question, person in my head!

The connection here is that as a software tester, if you log a bug without stating what you expected, you’re putting whoever is tasked with either fixing the issue, or deciding whether or not it needs to be fixed, at a massive disadvantage.

Some bugs, such as those that produce error messages in the UI, may speak for themselves, but much of the time there is a lot more ambiguity than you might think. What you expect to happen is obvious to you, because expectation is based on experience and personal bias.

To remove the ambiguity, all you need to include is a line or two explaining what you expected to happen, and maybe even why you think it should behave that way.

**But testers don’t decide how the software should work; that’s for the product owner!**

I have heard this several times in my career and it’s a nonsense argument. ‘Expected Outcome’ does not equal ‘Correct Outcome’, and it shouldn’t be approached as such. When I tell someone what I expect to happen, I fully expect a response such as they disagree and why, or what they would expect to happen instead.

As with a user story, a [bug report](https://blog.testlodge.com/how-to-write-a-good-bug-report/) should be a placeholder for a conversation, not a standalone oracle to be followed to the word. If project managers, scrum masters, developers etc., are just taking bug reports as gospel, there are bigger problems going on.

**Come on now, does it really matter that much?**

An ambiguous bug report can often result in a software change that from the developers point of view fixes the bug, but from the testers point of view does not. This ends up wasting everyone’s time, as it needs both fixing and retesting twice.

Surely the fact that including an expected result can minimise such waste is more than enough reason to include just a line or two in your bug reports?

Now that’s sorted out, all you have to do is sit back and work out what you’re going to do with the time and money you’ll save. I hear Antonio is available…

Diplomacy in Testing

Early in my career, there was a line I’d use in job interviews that the interviewers seemed to respond to pretty well; “If I’m having a good day, the developer is probably having a bad day.”

Some would laugh, some would nod, but oddly no-one ever pointed out that actually, it was a horrible statement. I cringe when I look back to those interviews. Not just at my dumb statement, but also at how people ate it up.



No-one does their best work out of fear of criticism. People do their best work so they can be proud of it, to feel like their hard work was worth it, and that people appreciate the effort they put in. When I’m writing one of these blog posts, I’m not focusing on the errors that Jane, the proof-reader might find; I focus on the experience of the reader at the other end. If, after I had submitted my first article, she had berated me for every little spelling and grammar mistake, my focus would have been on avoiding the criticism, rather than writing something worth reading. In fact, had that been the start of my writing career, I can’t imagine I’d still be writing these blogs at all. The reason why Jane knows not to take such a negative approach is because we’re trying to achieve the same goal, which is to produce an informative, and hopefully entertaining, piece of writing. In the same vein, a tester and a developer are also trying to achieve the same goal; to produce a good piece of software that’s a pleasure for the user to use.

**Defeat a Toxic Dynamic**

Historically, there has been a culture where developers and testers work as opponents, using terms such as “chuck it back over the fence” as if we’re feuding neighbours rather than teammates working to the same objectives. The origins of this toxic dynamic potentially come from methodologies like Waterfall where the development and testing of the software are cleanly partitioned. Meaning, that when the tester is looking at some functionality, the developer that wrote the code has moved on to other things.

This means that when bugs are inevitably raised, you’re forcing the developer to go back to something they may not have worked on in weeks, maybe even months. Imagine how frustrating that’d be if you’re in the middle of something new and you’re in the zone making good progress, then you get a notification that there is a problem with the code you wrote ages ago and now you have to stop what you’re doing to fix it. On top of maybe not knowing the code as well as you did, you might not be as familiar with the requirements now or remember the reasons for making the decisions you did. This makes talking about the bugs more difficult too and raising tensions further. In a perfect Agile world, this shouldn’t be an issue, but let’s be honest here; the perfect Agile world doesn’t exist. It’s often not possible, or even practical to test something immediately after, or even during its development.

**Talk To Each Other**

So if this is the problem, what is the solution? In my opinion, it’s one word: communication. Yup, surprise surprise in a blog about diplomacy, I’m suggesting the solution is to talk to each other!

For every user story or requirement you pick up to test, just talk to the developer about it. Ask them how it’s gone, how they’re feeling about it, and ask them to show you the work they’ve done. Compliment them on the things they’ve done well, ask questions about the technology they’ve used and the decisions they’ve made. Ask if there are any areas they feel they’ve had the most problems with, or are the highest risk. Then when you find a bug, talk to them about that too. The best-written bug report in the world can’t compete with a quick conversation, and maybe a demonstration of what’s happening, what you were expecting and why you were expecting it.

Same with if there is a comment or an email that you don’t agree with or understand. Have an actual conversation with the person rather than falling into a game of Jira (other services are available) tennis. Don’t fall into the trap of going back and forth endlessly when a face-to-face (or video call if you’re not in the same building) conversation could clarify the situation much faster.

**Build Bonds**

Doing all of this not only saves time, but also helps to build a bond and understanding between yourself the developers you work with. Doing this will reduce so much of the tension that can often occur, while also likely resulting in a better end product.

Diplomacy isn’t just about being nice; it’s about developing stable working relationships and shared priorities. It’s about respecting each other’s work and ideas and being able to put egos to one side for the benefit of the bigger project.

In a nutshell, it’s people being respectful to each other, being nice to each other, and generally just being decent folk.

In essence, can’t we all just get along?

Building a QA Team From Scratch

Following on from my previous blog “[6 Ways to Drive your QA Career Forward](https://blog.testlodge.com/drive-your-qa-career-forward/)”, we’re going to explore an area that doesn’t usually get much coverage because nobody can teach you exactly how to build a QA team from scratch. At some point in your QA career, you might find yourself moving into a fresh position as the sole tester in a company.

Initially, you’ll take all the knowledge you have into this new role and set up your [test cases](https://blog.testlodge.com/test-cases/), documentation, get to know your software and implement the company’s testing strategy. This will be something you will already know how to do at this point.



But there will come a time where your workload will increase due to the company’s expansion, the team will need to expand and you’ll find yourself in uncharted territories. For when that time comes, here are some tips to help you.

**What do you need?**

When it comes to bringing in more testers, what exactly do you need from your team? There are a few options to consider. Will you need someone experienced in automated, manual or performance testing? What combinations of skills do you require? Would it be worthwhile looking for someone with skills you don’t have so you can share your knowledge between each other. It could be a good opportunity to develop the team’s knowledge as well as your own.

In this day and age, remote working is also a valid option. It could be that your ideal candidate lives too far away to travel but have the exact skills you require. The disadvantage of this of course, will be the communication between you and them, and whether you can trust that the work will get done. This situation contrasts with that of on-site workers, where the communication and knowledge sharing whilst physically being able to see each other’s work are definite pros.

If it comes to discounting the perhaps perfect but remote candidate in preference to employing locally, do you have the time to train someone up with less or no experience and mould them into the tester you require? It’s unnecessary to always discount non-experienced people because you’ll find that some of them do have the right character and attitude to succeed in QA.

These are the initial questions you’ll have to ask yourself before moving forward in the recruitment process. This and the salary expectations will be factors in the initial process of finding a new tester.

**Get actively involved with recruitment**

The new job opening will usually be first offered to internal applicants for a set amount of time. You might discover a hidden gem within the organisation who wishes to move into QA. Some might even need a gentle push to apply for the role, as due to their lack of experience, they will have already discounted themselves. If no-one suitable steps up from within the company, the next step is to look externally.

Amongst the applications from external candidates that will arrive at your inbox, there many be some from an organisation’s HR or a recruitment agency. Either way, get involved in speaking to both of these. It’s the only way they will know your requirements for the role. Provide them with a job spec because this way they can filter candidates and leave you with a better quality handful to select for interview.

**Interview stage**

With the candidates now having been selected for interview, how prepared are you? It might be your first time on the other side of the table so make sure you prepare. You’re the company’s resident QA so you will need to know your role inside and out, as well as theirs. As this is likely to be your one and only chance to see if they have the right QA skills you require, ask them questions about QA, find out what they know and have used, and are their skills transferable? A tester needs to be proactive and think on their feet, so throw a spanner in the works by asking a completely random question in the middle of the interview. This is a good way to see if they can think quickly and outside of the box, which a tester needs to be able to do. They won’t be expecting it and it’s not something you can ever prepare for either.

Perhaps prepare a test for the interviewee and give a set time for its completion. A good place to start might be ISEB style questions (Information Systems Examination Board). You may also come across BCS Professional Certification via the same board. ISTQB (International Software Testing Qualifications Board) might be another option and are readily available online. This will test their knowledge on testing itself. You could also consider giving them a screenshot of the software you’re currently using. This way you can see how they would go about testing it without being given much specification to go with it. See how they formulate creating a bug. All of this will help you make your final decision and, if you decide upon that candidate, how much training they might need. As they will need to understand what testing actually is, a good way to find out is to ask the question “How would you test this kitchen appliance?” Offer them an appliance of your choice. You’ll be surprised at how many people completely overlook reading the specifications or manual. Reading available information is a fundamental part of testing software. Without doing this, how would we know what to test in the first instance?

It’s also an idea to take someone with you into the interview who may have more knowledge about the company as a whole. Select someone you can trust and whose opinion you value. After the interview has concluded, you will have the opportunity to discuss the suitability of the candidate with the colleague. Understand that people will be quite nervous in an interview so try to look past this and see how they coped with the test, with the questions you asked and with those they asked you.

Finally, ask yourself are they are the right cultural fit for the company. How do you think they will get on with the other business stakeholders?

**Make that call**

It’s now decision time. It could be that the choice is an easy one to make or on the other hand, it could be too close to call. To help narrow things down, look at factors like will they fit into your team and work well with the developers? Are they perhaps overqualified for the role, meaning are they just using it as a stepping stone? You can usually gauge from the interview if someone is likely to get bored in a role.

If after all of these questions you still find yourself struggling, go with your instinct and make that decision. Welcome on board!

**Be a manager**

You’ve now started building a QA team from scratch, but building doesn’t mean just bringing people in. The processes will be ever changing and it will now be your role, not only to manage and resource your testing team but look for new testing tools that work for your team. What used to work for you as an individual might not necessarily work for a team. Test management tools such as [TestLodge](https://www.testlodge.com/), which can manage your test cases, projects and test documentation is easily accessible online for you and your team to keep track of work.

Make it your business to know why something isn’t working correctly ,as no doubt you will be asked in meetings or stand-ups. It’s your responsibility to keep your team happy. Rotate them if possible, around different software projects at convenient times. This will not only keep them fresh, it will increase their knowledge in other pieces of software the company are working on. Offer them incentives of self development. You could even give them some time in the week to pick up new relevant technologies that will help them achieve both theirs and the company’s goals.

**Build your Empire**

Lastly, as you are now their manager they will look to you to celebrate them when they do a good job and defend them if something doesn’t go according to plan as you nurture their growth. You will pick things up as you go along. Sometimes you might struggle with some process or some implementation of testing tools, or perhaps difficult colleagues but in the end it will be something to be proud of, and after all you’ve built this QA team from scratch.

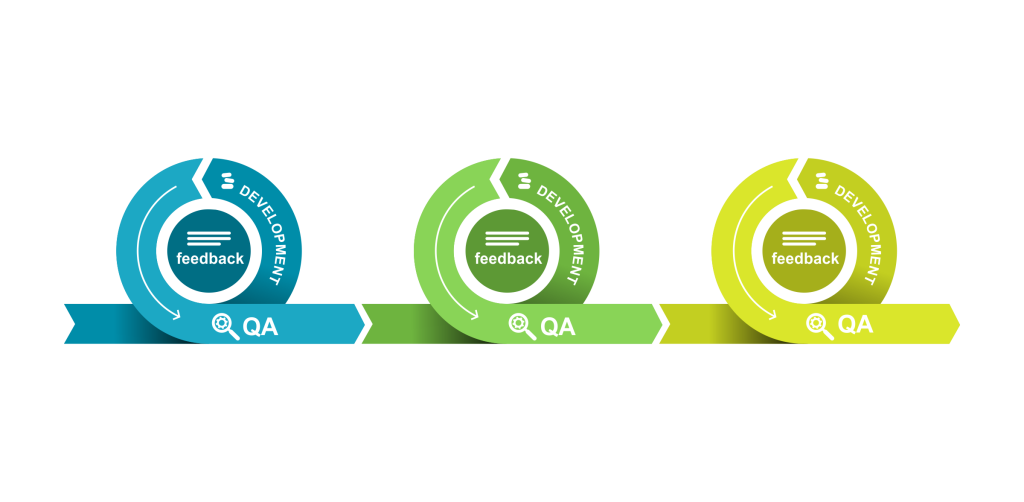
QA AND DEVELOPMENT: WORKING IN PARALLEL

In many software development teams, it’s common for QA and development to work independently. This is the standard way of doing things (at least it used to be). Developers would work on new features and pass them off to QA for testing. QA and development often wouldn’t even sit in the same part of the room, rarely interacting with each other. However, in agile software development, where small teams are working fast and releasing new features frequently, this isn’t the best approach. To be truly agile and to deliver a higher quality product in less time, QA and developers should be in sync, working together in parallel.

With agile teams being so quick to design, develop, and ship new features, QA teams need to work alongside the development team so they can understand the requirements and functionality of the software. Developers and QA need to discuss the best testing strategy for the code that has been written for the application. This ensures QA is up to speed on how the feature is being built, and is well informed of the required functionality. This also allows QA to be ready for changes that are bound to happen – changes to code, requirements and product decisions.

Testers are excellent at providing feedback. By integrating QA into the development process, you shorten the feedback loop. While the nature of QA is to find what is wrong with software, a true QA analyst inherently knows what’s right. Testers aren’t only looking for what doesn’t work; they are aware of missing features and have a strong sense of quality all around. In the same iteration, developers can make changes to a feature and give those changes directly back to QA for further testing.

So how do you balance the unique and often sensitive dynamic between testers and developers? After all, a tester is out to find things the developer did wrong or did not complete. QA see it as a success when they find a bug, but a developer sees it as a failure. QA jumping down the development team’s throat isn’t healthy. That’s demoralizing and not appropriate. Like any relationship, balance is key. Define a process, leverage tools, and most importantly, play nice. While QA and development have different roles, the goal is the same: to provide quality software to happy customers.



By integrating QA and development, both roles will learn and better understand the value they bring to the [software development life cycle](https://blog.testlodge.com/software-development-life-cycle/), introducing a new level of respect. Most importantly, bringing QA closer to development will ensure your team is able to adapt quickly, and provide a higher quality product to your customers.

QA and Support – Working Together

QA and support are very different roles. But in many organizations, it’s not uncommon for QA and support to work together in some capacity. In fact, in smaller companies you might even find one person responsible for testing *and* support. This doesn’t scale very well, but I’ve definitely seen it happen. In fact, I’ve been that person.



There’s actually a direct correlation between QA and support. If things aren’t tested properly before they get released, customers will contact support to report bugs. When support becomes aware of issues, they might go to the QA team and ask them what’s up… “How’d this bug slip through?”.

**When Support *is* QA**

When Support is responsible for QA (and vice versa), there’s an immediate incentive to test thoroughly. For example, if I’m responsible for testing *and*support, you can bet I’m going to do my very best to ensure no major usability issues slip through the cracks.

**QA and Support – Agile vs Waterfall**

Traditionally, support and QA are separate roles. Also traditionally, there are often “walls” between the QA team and the rest of the company. This barrier, often referenced as “throwing code over the wall”, leads to silo-ed information. While you can get away with this using the waterfall methodology, this lack of cross-collaboration and communication is detrimental to agile teams. This, in fact, is not agile at all.

In agile settings, support and QA need to talk to each other. Here’s why/how.

**1. The QA team can conduct feature handover for support.**

When agile teams have a dedicated QA team, they are often the last ones to see the feature before it goes live. But that shouldn’t be the case. Releasing a feature without looping in the support team is like . This is where a [support handover document](https://blog.testlodge.com/support-handover-document/) can be valuable.

**2. QA and support can do paired testing.**

[Pair testing](https://blog.testlodge.com/what-is-pair-testing/) can often replace the support handover document for smaller features. Sometimes a handover document might not be necessary for small changes.

**3. Sometimes support needs to test hot-fixes.**

By nature, support is very reactionary. They are constantly putting out fires. Sometimes this means dealing with bugs – and the angry customers who report them. This means support might take the role of QA for hot fixes.

**Closing Thoughts**

There’s a cultural advantage to QA and support having a good relationship. I’ve worked at companies where developers and QA didn’t talk to anyone else. You know what that is? A toxic culture.

QA and support working together promotes a healthy, collaborative culture.

Support Handover Document – Keeping Support In The Loop

Knowledge transfer plays a big role in the success of software teams. In software testing, knowledge is transferred *to* the testing team during the requirements analysis phase of the [software testing life cycle](https://blog.testlodge.com/software-testing-life-cycle/). Once testing is complete, knowledge is then transferred *from* the testing team to other involved people.



When testing is complete and the process is moving to production, your customer support team will start receiving questions from users about the new features/changes. It’s important that they know what’s going on because they are the ones in regular, direct contact with customers, answering product questions, writing documentation, and training users.

When your support team isn’t up to speed on what’s been changed, your customers are going to suffer.

For small teams, this can be pretty easy to manage. Your support team (or person) might sit next to the development team, and therefore have an ear to the ground on what’s going on. But in larger organizations, your support team might be on the other side of the room, or in a completely different country.

If this is the case, a Support Handover Document could be useful.

**What Is The Purpose Of A Support Handover Document?**

It happens all too often… testing is finished and a new feature is moved into production. Congratulations! But hold up, you missed one really important step; your customer support team doesn’t know how to use the feature. Even worse, they don’t even know the feature exists.

By staying in the loop, your customer support team is able to provide the best support possible at all times. A support handover document gives them the information needed to understand changes to the system and answer questions about the said changes.

What’s changing? When is it changing? How does it work? Any risks or issues to be aware of? All of these things need to be communicated to your customer-facing folks. A support handover document captures all of this information in one place.

**What Should Be Included In A Support Handover Document?**

**Release Date:**The support team needs to know *when* something is going to be released so they can prepare ahead of time and be ready to support it.

**Summary of Changes:**A definition of what has been changed and why? What extra value are the changes expected to provide the users? Support needs to know these things so they can speak intelligently around the changes and add context when necessary.

**Known Issues and Workarounds:**If there are any possible issues users could encounter, they should be documented here. Are there any workarounds? Support teams should be armed with the necessary tools to help the user successfully.

**Usage Instructions:**In general, how does this thing work? Your support team or documentation team is going to need to write product help documents. Give a quick rundown of how the new thing works. Maybe this means including user stories in the doc so they can run wild in the testing environment. [Exploratory testing](https://blog.testlodge.com/how-exploratory-testing-helps-customer-support/) often comes into play here.

**Instructions for Troubleshooting:**If something goes wrong with the new feature, what steps should be taken to determine why this happened, and how it can be resolved? This might be referencing some logs or asking specific questions. Support needs to be able to solve these types of problems.

The nature of support is solving problems and answering questions. Sometimes support reps are going to hit roadblocks, but it helps to be as prepared as possible and have a good understanding of what’s being changed in the product. In a scenario where support discovers a critical defect, an issue should be logged internally and the development, or [devops](https://blog.testlodge.com/what-is-devops/) team, should be engaged.

**Fitting This Into Your Process**

Do you really need to create a support handover document? No. But if your support team is struggling to stay up to speed on product changes, it might be worth considering. So how do you squeeze this into your current development process?

I’ve seen handover documents come from testers and product managers both during and after testing. You might consider tagging issues in Jira (or whatever tool you use) as “needs support handoff” to keep track of which items require detailed communication to support. Or, once items get to a certain phase in the process, maybe you do a quick audit to see which items require a support handover doc.

Another way to transfer this knowledge is through a handover demo. Many teams have weekly demo meetings where new features, changes, or ideas are demonstrated. These meetings provide a tremendous opportunity to pass along knowledge to other teams such as support.

Bring your support team in early so that they know what’s coming down the pipeline.

**Closing Thoughts**

There’s no set way to create a support handover document. The key is to make sure the necessary information is shared and communicated to the right people *at the right time*.

How Exploratory Testing Can Help Customer Support

We’ve talked about [exploratory testing](https://blog.testlodge.com/what-is-exploratory-testing/) in the past… how it involves little planning, a bit of scientific thinking, and it’s fun. But who actually does exploratory testing?

Chances are, if you’re a part of a software team, you’ve already done exploratory testing yourself. Maybe you’re a product manager trying out a new build, or a marketing person testing out a new feature for a blog. Or, maybe you’re on the support team trying to figure out how something works.

If you find yourself in any of these situations, you’re likely not following any test scripts… you’re doing exploratory testing.



Exploratory testing is valuable not only for testing, but also for learning about the product and it’s features. Support teams are often left to “explore” the application and figure out how things work, because, lets face it, in order to support a product, you need to know how to use it.

Ideally, you learn how to use new features *before* you get questions about them. This is where exploratory testing comes into play with support teams.

**Bringing Support Into The Mix**

It’s important to keep your support team in the loop on new features, fixes, etc. It’s equally important for your support team to make sure they’re kept in the loop on these things.

Set up some sort of feature handoff from QA/development to support. Depending on your development process, this might mean support pairing with someone to learn about the product. Or maybe it means new features/issues in your product management tool are tagged with “needs documentation” so that the support team (or whoever is writing your product documentation) knows something requires new or updated documentation.

Whatever it is, support teams need to know what’s changing in the product. If you’re on a product team and this isn’t happening already, you should work to implement a change.

**Advantages Of Bring Support Into The Mix**

Looping in support on new features doesn’t just help the support team learn about new things, they can also save the day by finding things that may have slipped through the crack and didn’t get caught by development, testing etc. Support is at the front lines and therefore they know and understand the customers.

When support is aware of changes and understands them, they aren’t caught off guard or forced to escalate things/delay responses when questions on those changes come in. This closes the loop between product development and the customers.

**Why Support Should Do Exploratory Testing**

The best way for a support person to learn and understand the product is to use it. Exploratory testing can be used to help answer the questions “how does this work?” and “what happens if I do this?”. When support can answer these questions, they are better suited to be answering questions from customers who encounter these areas of the product.

It can be cost effective and efficient for support to do some quick exploratory testing on new features and changes in the product. It’s not really necessary for them to follow test scripts when learning about the new product or testing things out. In fact, support people might approach testing differently than a QA team, so giving them the slack to truly “explore” the product can be to your advantage.

**Conclusion**

Exploratory testing helps support teams understand how things in the product work. It can also provide a final opportunity to find major usability flaws or bugs in the system. This type of testing is a low cost solution for finding bugs and truly “exploring” the functionality of software.

What is Pair Testing?

Sometimes two is better than one.  In pair testing, two people sit behind one machine to test the software.  One person controls the mouse and keyboard.  The other person asks questions, discusses test scenarios, and takes notes.  Pair testing is a type of exploratory testing, where two team members discuss and share ideas and scenarios around the software.

Pair testing is a collaborative effort, versus a single-person testing effort.  Typically, one of the team members is a tester and the other is either a developer or a business analyst.



Pair testing is not meant for executing formal test cases and should never replace other types of testing.  It often works well in agile environments as a supplementary part of the overall development and QA process.

**How to do Pair Testing**

**Pair with Someone**

The first requirement is finding two people who can work well together.  If there are personality clashes with the two team members, pair testing will likely not work.  The team members must be able to communicate with each other and share the goal of increasing the overall quality of the software.

**Prepare the Environment**

It’s important to be prepared and have the appropriate environment setup.  Find a quiet room with enough space at the desk for two people, avoid distractions, and try to commit 1-2 hours to the session.

**Determine the Goals**

Pair testing should not be used to execute formal test cases, but it’s a good idea to come prepared with a simple list of ideas and areas you want to cover.  You could be testing a new feature, or multiple components… just have some sort of idea on what you would like to accomplish.

**Determine the Roles**

Determine who will be controlling the keyboard and mouse; this person is called the “driver”.  The other team member, the “navigator”, will help guide the session, think out-loud, and suggest certain scenarios.  Once you’ve completed the session and you have your notes, log the bugs and feedback.

**Take Notes**

Bugs shouldn’t be logged during the testing session – wait until the session is over.  Use the testing time to discuss ideas, scenarios, and to try different things.  And remember, once the session is over, take the time to log well-written bug reports.  Your team will appreciate it.

**The Advantages of Pair Testing**

While pair testing might not work with every team, there are several advantages to this type of testing:

1. **It helps transfer knowledge.**New information is shared and new perspectives are discovered.
2. **It often promotes negative testing.**  It’s not uncommon for software developers to code for the happy path and not focus as much on the negative scenarios.  Pair testing ensures more angles are tested.
3. **When identifying bugs, the information is validated by two people.**  This ensures that when bugs are logged, they are logged accurately and with the necessary information.
4. **It can save time.**  In a pair testing session with a developer, the developer might be able to point out the cause of bugs quicker since they are familiar with the coding and framework.
5. **Pair testing is fun, helps build culture, and breaks down barriers.**  When two people work together, it can lead to a better understanding of each other.  This creates a stronger working environment and both team members gain an understanding of the complexities of the software.
6. **Pair testing improves communication.**  Perhaps the most important part of business, communication skills are honed as two people work through challenges together.
7. **It breeds positive energy.**  As long as the two people get along, pair testing can often lead to new, positive energy.  Working together to solve problems creates a team atmosphere.
8. **Pair testing can be used as an opportunity to train other employees.**  For example, customer support can be the “navigator” while the tester “drives” – this allows customer support to understand the workings of a feature and can also help drive good support documentation for customers.

Pair testing is a unique approach to software testing.  Different than most other types of software testing, two people work together to explore a feature or component and the software and the team are ultimately affected in positive ways.

Developers And Support Should Be Friends

There’s nothing worse than being afraid of someone on your team. You know… that guy who’s really intimidating, doesn’t *really*listen to you because you’re not a developer (maybe you’re QA or support) and you’re not on his “level”.

Yuck. These situations create a toxic environment and quickly deteriorate company culture.



In this post we’re going to talk about how the development team and customer support team should be friends, and how when these relationships are strong, good things happen.

**Understanding Boundaries**

First, it’s important to understand boundaries. As someone on the customer support team, you don’t want to be throwing bugs to development left and right and distracting them every time you do so. Developers need uninterrupted time. It’s important to respect that. Devise a plan to channel information to them.

Make sure you have [a process that works for both sides](https://blog.testlodge.com/enjoy-the-process/). If you’re causing a commotion every time a new bug comes in, you’re going to get less people listening to you when those high impact issues surface. By having a clear understanding of these boundaries, your customer support team is sure to get respect from the development team.

**Gathering All Of The Right Information**

But it doesn’t stop here. When issues arise, support needs to bring fully fleshed out issues to the development team.

“There is a bug”. That’s good to know, but it’s not very helpful. Do some of the leg work and find out as much information as you can. How many users does it affect? Which browsers and systems? Where exactly is the user getting stuck? Are they really upset or just sharing feedback?

After you get all of that information, compile it in a way your development team can quickly understand. Make it easy for them. Be careful with suggesting solutions though – that’s the product team’s job.

**Breaking Down The Walls**

Understanding boundaries and getting all of the right information to the development team is a great start to building good relationships between support and development. There are also things you can do that aren’t directly business related to get closer with your coworkers.

When is the last time you your development team and support team got lunch together or did happy hour? Bring a little fun into the mix. Take the time to get out of the office and interact without business being the main topic of discussion. Getting to know your team at a personal level can really help build trust and respect.

**Advantages Of Strong Relationships Between Support And Development**

Lets take a look at some of the advantages you’ll see when support and development have a strong relationship.

**Bugs Get Fixed Faster**

When there is mutual respect between development and support, the development team is more likely to prioritize issues your support team is reporting and do “favors”. Lets look at this from the support rep’s shoes…

It’s easier to ask someone to do something for you when that person likes you and understands why you’re asking them to do that. It’s much harder to ask a developer to do something when they don’t respect you.

**Builds Empathy For The Customer**

When the development team makes an effort to understand the work support is doing, they have a better chance of building empathy for the customer. Understanding the customer is important. With the customer at the top of mind when developing new features and fixing bugs, your product will be more user-friendly and solve the problems your customers are actually dealing with.

Some teams even practice “all-hands-support” which allows developers to talk directly with customers around product feedback, issues, and basic product questions.

**Internal Tools Get Focus**

Building internal tools can really benefit the support team *and* the development team. Here’s an example…

Perhaps you’re seeing a lot of support requests from customers wanting to increase the amount of team members they’re paying for. This has always been done in the database – a developer would have to go in and change the maximum number of team members aloud on a plan and then let support know the change was made. This takes away time from the development team and also leads to longer resolution times (because support had to wait for the developer to complete the task).

Eventually, support asks the development team to give them the ability to do this so they can stop bothering them and complete these tasks themselves. The result is that by spending a small amount of time on an internal tool, development was no longer bothered by these minor requests and support could complete them on their own. This also lead to faster resolution times and happier customers.

**Closing Thoughts**

When developers and support work closely together, the whole team *and* your customers benefits. The quality of your product improves, and your customers are happier with faster bug fixes and shorter resolution times. It’s important to identify any risks that could lead to broken relationships between support and development and work to eliminate them sooner than later. A good culture starts with strong relationships internally.

Regular One-to-One Meetings: Do You Need Them?

Do you have regular one-to-one meetings at work? Perhaps you’ve tried them but ended up considering them ineffective. Despite the ‘Meetings suck’ and ‘Meetings, the alternative to work’ stuff all around, most experts recommend having regular one-to-ones between employees and their managers. If these meetings haven’t worked for you, then most probably, some common mistakes have been made that have prevented you from getting tangible results. Let’s figure out why.

****

**Why Have One-to-Ones?**

Regular performance review sessions can be disappointing. You seemed to have delivered great results but during the meeting but you find out your boss is not so happy.

Or perhaps you are a QA leader, and here comes your subordinate stating that he is leaving for another company… but you had no idea!

**Why do things like this happen? Turns out, it’s all about expectations.**

Attending you quarterly KPI (key performance indicators) meeting, you expect the usual acknowledgement and pay raise as a result of hitting the goals of your previous performance review. Meanwhile, your manager also comes with a list of expectations, met or not. Along with your achievements, he brings up your failures like constantly being late, inaccuracy in describing the bugs, or your low performance. Until that point you didn’t challenge his observations, and weren’t even aware of them. And that’s when you find yourself frustrated and undervalued.

With regular one-to-one meetings, you can synchronize mutual expectations and discuss any updates. You can give and get timely feedback, so that when you have your next KPI review, no one is that much surprised by the agenda. As a manager, if you want to stay on top (as well as build clear and easy relations with your team members), you should definitely take advantage of having one-to-ones.

**How to Hold One-to-One Meetings?**

If you want to ask your manager for a meeting, then do it. Don’t be afraid or feel like you are stealing their time. As long as you have a list of relevant questions to discuss, it’s not a useless activity. Your initiative and responsibility in terms of your work and career, will more likely give you bonus points in the eyes of your manager.

If you are a manager and want to give one-to-ones a try, explain the purpose to your colleagues in advance, and let them know how they can benefit. From previous experience, they may be embarrassed seeing just another sudden meeting invitation, or would be expecting themselves to get chewed for some unknown reason. Anyway, their attitude won’t help in your efforts to set a positive attitude and rapport.

Below is a list of items for an agenda that you could use on your weekly (or bi-weekly) one-to-one meetings.

As a manager:

1. Ask your colleague what they would like to discuss
2. Inquire about their concerns, anything that bothers them in terms of the project, general working conditions, relations with the team members/clients, or anything else?
3. If you’ve had previous meetings, check the status on any actioned items.
4. Give your feedback if needed.
5. List new or updated action items based on your conversation. Specify who should do it and when.
6. Share the meeting minutes with your colleague. As my boss used to say, no meeting minutes means no meeting held.

Listen, listen and listen actively, and take notes. Then don’t forget about your commitments. When your subordinates see that action items are not just words on paper, that you hear their concerns and get your promises done – they will know one-to-ones work, and most likely grant you their loyalty and respect.

As a subordinate:

1. Ask for feedback
2. Bring up your questions or concerns. If you want to discuss long-term goals, ask what you can do for the company or project to achieve them. Be specific: “What do I need to deliver to gain the Senior QA position in 1 year?”
3. These questions will get you a list of action items (with Who/What/When points) that will become a baseline for your further meetings. Suggest scheduling the next meeting in 1 or 2 weeks with the intent of making them regular.

Try to be flexible. You can approach different people differently if you know (or guess) their psychological type. Some people love figures or visualizations; others are more concerned with people and relations. Some need to be given short meeting agenda in advance; others are fine being caught for an occasional meeting during the lunch. For example, you may find it interesting to learn about four [management styles by I. Adizes](http://adizes.com/management_styles/) which can help when you want to approach your manager and other colleagues, regardless of the position you take.

And one more thing, be sure to be natural in your approach. Don’t use a tone or phrases that are not inherent to you or might be irrelevant to your working environment. People notice things like that, so just be natural.

Of course, one-to-one meetings work best at companies with units or departments, where a manager has 3 to 10 subordinates and can actually affect their career and work conditions. If you are in charge of more people, delegate such activities to your direct subordinates such as team leaders, resource managers or whoever is close to the team members and hold one-to-ones with them, in turn.

It may be more difficult to use one-to-ones in a distributed team. Video calls help but are not as good as live communication. If you have the possibility to meet with key remote members in person (in addition to regular calls) – do it. You can also use one-to-ones with your clients, if you are in a relevant position such as a freelancer QA leader or manager.

**Finally**

Introducing one-to-ones may feel awkward at the start, but as long as you find your own ways to make them work, and build trustworthy relationships and rapport, you will feel all the benefits of regular meetings, whatever position you hold.

Good luck in your career, and be inspired!

Characteristics Of A Good Software Tester

Software testing isn’t for everyone. A top-notch engineer might be a weak tester. A seasoned project manager might have the skills to really drive a project through to completion, but when it comes to testing, they just don’t have it. Like all roles, there are certain characteristics of a good software tester.



**Top 13 Software Tester Characteristics:**

**A good software tester asks questions**

Software testing requires asking a lot of questions. “What happens if \_\_\_\_?”, “Why isn’t \_\_\_\_ working?”, “Why does it work like *this*?”. Questions inspire new test cases and different ways of approaching the tests. When QA is involved early on, questions are asked early on, and quality is injected early on.

**A good software tester is curious**

Good software testers want to know what’s going on behind the scenes. They’re constantly looking for problems to overcome. A curious software tester is more likely to find bugs and usability issues than a non-curious tester.

**A good software tester is a strong communicator**

Software testers are often the bearer of bad news. They need to be able to communicate effectively and passionately, but also delicately. From written communication to verbal communication, being able to communicate problems and weaknesses, steps to replicate bugs, and why something should be a certain way is an art form.

**A good software tester is patient**

It’s easy to get caught up in the excitement of finding a bug or identifying an opportunity for improvement. Software testers often get “no” as a response. Being patient is key to being a successful software tester and building strong relationships with your team.

**A good software tester is a strong writer**

Software testing involves a lot of writing. Writing test cases, writing bug reports, writing emails. Poor writing leads to communication breakdowns and wasted time, both things most software teams can’t afford. Writing efficiently and effectively goes a long ways in software testing.

**A good software tester meets deadlines**

There’s never enough time to test *everything*. The ability to prioritize things and stay on task is crucial for meeting deadlines. Time management is very important to being a good software tester.

**A good software tester is empathetic**

Just like a good designer should design with the user at the top of mind, testers should test with the user at the top of their mind. Putting yourself in the shoes of the user will help uncover different problems and areas for improvement.

**A good software tester thinks creatively**

As a software tester, you need to approach things from different angles. Think outside of the box and try different things. Try to break it, and try to break it in different ways.

**A good software tester is organized**

Organization is crucial to being a successful software tester. As you know, testing involves a lot of documentation, artifacts, time lines, and communication. If you drop the ball, it affects the entire team and project. Use the right tools and processes that work for you and your team to stay on top of the ball.

**A good software tester has technical skills**

Knowing which questions to ask requires some technical knowledge. Technical skills also help you understand limitations and boundaries within the application.

**A good software tester is a team player**

Be easy to work with. Being a team player encompasses many of the other qualities we’ve discussed. Remember to appreciate the work of your team. Developers have an entire different skill set than you and they *are* smart even if they are the source of bugs. When things go wrong, remember you’re on the same team.

**A good software tester is quality driven**

Quality is a mindset. A quality-driven person lives and breathes quality every day. As a software tester, this is where the passion for “quality” comes from. You strongly desire the best possible product and don’t compromise for quality ever.

**A good software tester pays attention to the details**

A good software tester is thorough. Testers need to drive things to completion without missing a beat. Remember, testing is the last phase of the [software development lifecycle](https://blog.testlodge.com/software-development-life-cycle/). Your team and your customers are relying on you.

**Conclusion**

As you can see, software testing takes a unique set of skills. Characteristics of a good software tester include both hard skills and soft skills. Testing isn’t for everyone. It takes a creative, technically-minded person to be a successful tester. Approach your job with these things in mind, and you’ll find yourself becoming a better tester over time.

How to be Creative in QA

In the world of software development, the dedicated software tester (also known as the QA engineer or QA analyst) plays an important role in bringing the product to market, but they don’t actually create any tangible part of the product. So, what is the QA Engineer’s creative input to the project? The UI/UX designer made the look and feel, the software developer wrote the code, but what did QA create? Well, the plain answer from an end user is absolutely nothing, even through the product would have been different without the contributions of QA. As a result, the software QA position might be perceived as the least creative contributor on the development team. In any case, from the outside, it looks like a low creativity position, and there are some valid reasons for thinking that.



The primary function of the software tester is to verify that software is working in the way it has been designed. In other words, the “creatives” envisioned it, designed it, and coded it before QA came in to verify that it meets predetermined criteria.

Does this mean software testing is a role that requires little or no creativity? Absolutely not. In fact, a creative mind is crucial to being a good tester.

As a long time software tester, I know from experience that I am much more satisfied in my work when I can exercise my creativity. It’s important for anyone in a software testing position to find ways to creatively make something that is valuable to the development process.

**4 Ways to Exercise your Creative Brain as a Software Tester**

**Design Contributions**

Over time, software testers become very familiar with a product’s design, functionality and potential uses. In fact, QA may eventually know the design better than the team who initially drafted the product’s concept. This inevitably happens because testers are acutely aware of every change made to an application’s workflow, even when the design team has moved on to the next project. In some cases, designers come back to previous features only to address issues. With their experience, a software tester becomes an excellent resource for the design of new features and functionality that will be integrated into an existing product.

**CREATE Test Cases**

How does testing uncover design flaws? UX/UI designers rely on QA to uncover flaws in the design of a product by quickly identifying scenarios where the design is not adequate. During the software development process, testers and developers are given requirements for features to be built. They include all the desired functionality for when a feature is implemented, but it doesn’t cover every scenario that should be tested. When developing use cases for testing, the tester must consider all possible uses for the feature. This requires the tester to think like an end user and produce scenarios that will test the design and the functionality. Creativity in the test case authoring for those use cases, is crucial to uncovering design flaws and other bugs during the development process. The tester must consider the uniqueness of each user and plan for that user’s individual approach to using the application.

**Test Result Reporting**

After test cases, the next most important aspect of software testing documentation is the reporting of test results. The first part of this documentation is the low-level reporting of defects, but reporting the discovery of defects should be considered the most important piece of work delivered to the development team. Without the proper steps to reproduce a defect and other vital information such as the environment under test, the team cannot capitalize on the excellent work done by the QA team.

Documenting these items provides an opportunity for QA to be creative. Making screenshots, videos and additional visuals to provide developers with the clearest information possible regarding defects is an opportunity for a tester to use their creative prowess.

At a higher level, test result reports provide a view of the state of the product. Statistics showing the percentage of test cases that have passed, failed or have not been executed, provide insights to the state of the application and guidance for projecting a completion date. Other stats such as the number of bugs fixed or re-opened during a test cycle, give management an idea of the levels of progress being made. The reporting of these numbers can be done in a variety of ways. This is an area where QA can creatively produce documentation that can reinforce the value of software testing and provide product managers with information necessary to make decisions tied to resources and timelines.

**The Importance of Documentation**

Because software testers don’t create the code that produces the end product, documentation delivered by the QA team is particularly important. Without documentation, it is difficult to quantify the work and value of QA.

Each piece of documentation requires thoughtfulness and creativity. The QA engineer should continuously strive to improve the effectiveness of documentation that is produced.

**Conclusion**

I have hopefully convinced you that it’s needful for software testers to exercise creativity so they can perform at a high level. Everyone has an innate desire to be creative. If you are in QA, try to document more, even if it might not seem necessary at the time. The creation of documentation or reports will give testers the satisfaction of making something that displays their effort and creativity. This goes a long way when their primary role is evaluating other’s creations.

Working Remotely: Advantages and Disadvantages

Today more than ever, people are making a living from there home. The need for a physical, dedicated office space is dwindling as new technology, tools, and workspaces emerge. People are finding new ways to make an income through creative services, software products, and more – they’re creating their own businesses. And many of these new businesses (especially technology-driven ones) aren’t investing in offices – they’re investing in people and operating 100% remote.



But working remotely isn’t for every one, and certainly not for every job type. You need to have the right team in place, and it takes practice. There are a number of advantages and disadvantages to working remotely. Lets take a look at some of them.

**Advantages of Working Remotely**

**Increased Flexibility**  
It’s no surprise that one of the biggest perks to working remotely is the flexibility to work wherever you want. This makes it easier to juggle work, family, and other obligations. Work life balance can improve greatly by working remotely – but be careful; without the right boundaries, the opposite can happen.

**More Productivity**  
When done right, working remotely can lead to a more focused workday. Employers fear the distractions of TV and household chores, but the office comes with noise, meetings, and many other interruptions from colleagues. Many people work best in their own space, with piece and quiet.

**Reduces Commuting Cost & Time**  
Working remotely allows employees to spend less time and money traveling to the office, which saves a lot of money in the long haul. This also means more time in the day to be productive.

**Saves on Office Expense**  
If you’re operating 100% remote, you’ll save a lot of money by not renting or owning an office building. Cha-ching!

**You Can Hire Anyone**  
Why limit yourself to hiring people within a certain geographical location? By hiring remote team members, you can find the best talent to get the job done, no matter where they’re located.

**24-Hour Support**  
Hopefully you have customers all around the world using your product at all times of the day. By hiring in different time zones, you can provide 24-hour customer support to ensure those customers are taken care of no matter what time it is.

**Disadvantages of Working Remotely**

**Company Culture Can Suffer**  
Say goodbye to NERF gun wars and ping pong tournaments with your colleagues. It can be hard to build a company culture remotely. You’ll need to find ways to keep your culture alive through technology and periodic face-to-face meetings.

**Communication & Collaboration Get Harder**  
Personal communication and collaboration are vital to a company’s success. If you don’t focus on this, it will hurt you. Email is a remote team’s communication killer. Use video chat, collaboration tools, and group chat to build a “virtual office”. Try new things until you find something that works.

**Domestic Distractions**  
Lets be honest, there are plenty of distractions at home too… the TV, kids, pets, chores etc. You must learn to balance these properly. Build trust with your employer by getting your work done, and being available when you’re expected to be available.

**Fewer Boundaries Between Work and Personal Life**  
While work/life balance can improve by working remotely, that line is also at risk of becoming a little blurry. Define boundaries (set a structure) and stick to it to avoid becoming a workaholic.

**Conclusion**

Working remotely takes practice, commitment, and strong communication skills. It’s not for everyone, but you’d be surprised at how many well-known brands are operating remotely today. After all, who doesn’t like a little freedom, trust and flexibility in their lives?

Q&A with Jon Lane – A Remote Tester

I met Jon Lane a few years ago at UserConf in New York City. At that time, he was part of the Harvest support team. He gave a presentation about how he was leveraging various tools and technologies to create an efficient customer support system. I was impressed with his ideas and could tell he had a knack for quality.

****

A year or so later, we crossed paths again in San Francisco. At that point, Jon was dabbling in QA and beginning to take the lead as Harvest’s first dedicated QA person. I caught up with Jon recently to talk about testing and working remotely.

**Q. What is your current role at Harvest?**

**A.** My official title is QA team lead; there are three of us in QA and I head things up there. My role involves coordinating with the product team on new launches, writing test cases, running those test cases, writing automated test cases, and then the managerial side of things that come with leading a team.

**Q. Where is your team located?**

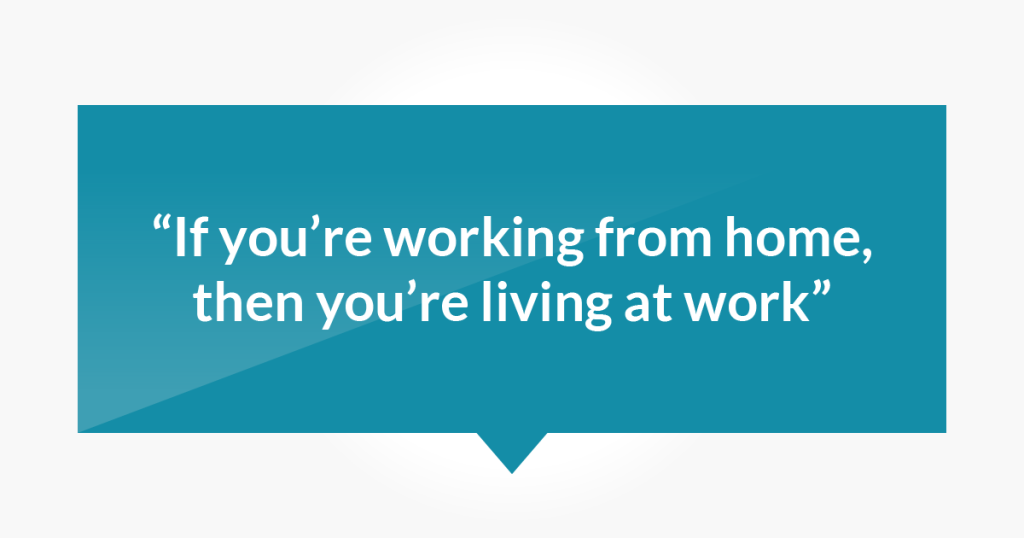
**A.** We’re a distributed team; one is in Chicago, another in Sofia, Bulgaria, and I live on Mayne Island, BC.

**Q. As a remote worker on a distributed team, what tools & processes are in place to help you integrate with the rest of the team?**

**A.** Well we use Trello, Slack for real-time communication and TestLodge for documenting tests. We use a little bit of Basecamp but not as much any more – we’ve mostly moved over to trello for that. We use Google hangouts for video conferences so that everyone can see each other’s faces. We also use Google docs for long form writing documents.

**Q. What challenges do you encounter as a remote worker?**

**A.** Harvest is a bit unique in that we’ve always had remote workers. The first person they hired was remote, so the company has always had that “remote” mentality.

****

If you’re working from home, then you’re living at work. Our main office is in New York and there’s a dozen or so people that work there, but the rest of us are distributed around the world. You feel a little bit disconnected sometimes when the New York team does fun things after hours because you don’t get to participate in those, but we try to help that by doing a couple company-wide meet-ups every year. Our next one is coming up in February – we’ll all fly into New York and get to spend a week together.

**Q. What do you like about working remotely?**

**A.** I don’t know that I could not work remotely any more to be honest. [laughs]

Having the flexibility to get up and start work in the morning, run my kids off to school, finish up work, pick up my kids from school in the afternoon, have time to cook and clean… I don’t know how I could do that if I had to commute to a job on a daily basis.

It’s afforded me a life that few people have. I live on an island with about a thousand people on it; there isn’t exactly a tech industry here. [laughs]

**Q. What development methodology does Harvest use and how is it working with a remote team?**

**A.** That’s a tricky question because we have a few different teams/products across Harvest.

We have the core Harvest app team, we have our mobile apps (Android, iPhone), desktop app, and we have Forecast, our project planning/resource allocation app that we launched about a year and a half ago – these are all separate teams and it’s really a little bit all over the board.

We’re not really strict on one methodology. Basically what we’ve done with the QA team is developed our own structure for how we work. It involves getting us the requirements, a Trello card, and then we’ll work with each team within our framework for managing things.

**Q. In what capacity do you work with the development team?**

**A.** Me personally, I spend probably half of my time working with the product teams – testing, reproducing bugs, reporting etc. The other half I spend on our automated test efforts as well as the management overhead stuff.

Test cases probably take the most time – making sure you’re writing good test cases takes time. Actually doing the tests doesn’t take all that long and chances are I’ve already run the test several times while writing the test cases, so you kind of know what to expect already.

The back and forth between me and the development team isn’t too onerous because we can link directly to one of the test cases we’ve written, this give the developers all the steps needed to reproduce the issue – so it’s pretty easy to reproduce something once it’s been found.

**Q. Are you re-using test cases?**

Yea, they don’t get forgotten about. Most of the test cases we write will be used for regression testing down the road. So when some features are changed or updated, we’ll be able to go back to those test cases and run them again.

**Q. Do you ever find that writing test cases is almost too laborious for the value you get out of it?**

**A.** It seemed that way starting out, but for us, test cases are almost a way of communicating with the product teams. It gives us accountability to them and shows them what we’ve done in terms of our testing.



Before, they could work on something and hand it over to QA, but that doesn’t provide the product team a lot of confidence in the test. Once you’ve documented individual test cases describing exactly what/how you’re testing, it gives the product team a higher level of confidence that we have good test coverage.

We had an incident last year where a bug slipped through into production and we were able to go back into our test cases and ask “did we have a test case for that?”, “was it giving us different results?” or “what happened there?”. In this case we found it to be something that was missed in the requirements and therefore a test case didn’t get written around it, which lead to the bug slipping through.

When I started at Harvest we didn’t have a formal QA department, people would kind of just do QA on an adhoc basis. Some people see the value of qa more than others, and writing test cases and formalizing the process really helped gain the trust and buy-in from everybody. It was probably the biggest thing that helped solidify our importance to the development team.

**Q. How much automated testing do you do?**

**A.** We went with a 3rd party solution for automated testing in 2015 and then they wound down their operations at the end of the year so we lost a good amount of work on automated testing when that happened. In 2016 we’re going to be re-evaluating that and very likely getting more into automated testing in-house using selenium. We’re already using selenium a bit and will probably be leveraging that more so in 2016.

**Q. How much manual testing do you do?**

A. In general I’d say 75% of our testing is manual and 25% is automated. There isn’t a lot of new features that we’re testing in an automated way before it hits production, we look at automated testing as a safety net.

**Q. How do you use TestLodge?**

A. We’ve actually expanded our use of TestLodge within the last year. Our general work flow is when the product team finishes work on something, we have them use the requirements section in TestLodge to give us a description of what the new functionality is, what things should happen, what things shouldn’t happen, and that kind of thing.

They (product) will then create a card in Trello pointing to the requirements in TestLodge and that sort of kicks off the testing process for us. The product team will also include any miscellaneous information we might need to know such as what staging environment the feature is in etc. We’ll then take those requirements and turn them into one or more test cases per requirement and we’ll execute the test runs from there.

Once we finish our test runs, we’ll link back to the test report in TestLodge showing any failed, skipped, and passed tests. We also use the Github/TestLodge integration because we use Github for issue tracking. So anytime a test case fails, it automatically creates an issue in Github for our development team to look at.

**Q. Any advice for someone new to a testing role?**

I started out in support at Harvest and happened to be more technically inclined so I was handling more of the technical questions that came in. You need to have a bit of a technical mind.

Mainly, internally we look for in-depth product knowledge. I had been here a couple years before I moved into QA so I already knew the ins and outs of where things were likely to break, the rough edges of the product etc. So by the time I made the move into QA I was pretty well versed on the product.

In general, every new hire at Harvest does about a month long support rotation answering support tickets – that really helps to build up product knowledge. Just hearing about the problems people are having with the product, clicking around and trying things for yourself.

**Q. Any advice for other remote workers?**

For your own sanity, try to stay connected, not only to your company but to the community as a whole. I’m on a really small island (literally) but I’m pretty close to a few major cities. I try to get lunch with other people in those cities when I can. I also attend conferences and even online webinars.

Tips for Working Remotely

It takes a lot of dedication to work remotely successfully.  You need to be very motivated and committed to the job, not just the idea of remote working.  There are certain skills, tools, and best practices that can be implemented to make working remotely actually work.  Working remotely certainly isn’t meant for everyone.  It takes the right people to build a successful remote team, but it can have a number of advantages, which [we outlined in a previous post](https://blog.testlodge.com/working-remotely-advantages-and-disadvantages/).  In this post, we’re going to discuss some tips for getting the most out of working remotely.



**Create a Structure**

Structure is important in any effort to be a productive, efficient worker.  In many ways, you should treat working remotely as if you’re going to the office.  Set a schedule, wake up, shower, and get dressed just like you would if you’re going to the office.  Set goals for each day so you can track your success and productivity.  Dedicate a space with a good atmosphere to work.  A noisy background or poor internet connection can make working remotely very difficult or even impossible.

**Set Boundaries**

It’s important to separate work from home, especially when working remotely.  If you have kids at home, setting rules and expectations with them will help your workday go smoother and ensure you’re able to focus when you need to focus on work.  It’s important to take breaks and get out of the house too.  Just as you go for coffee runs at the office, take a little walk around the block – a little fresh air and exercise can go a long way.

**Be Organized**

As a remote worker, if you’re not organized, you’re going to have a hard time.  Without a colleague or boss next to you, you’ll need to prioritize your tasks and stay focused.  Use a checklist so you know what’s on your plate and don’t let things fall the through the cracks.  Keep close tabs on your calendar.  You need to build trust as a remote worker, and if you’re missing meetings or always late, that’s not going to look good.

**Communicate**

Communication is important in any work environment, but when you’re working remotely, strong communication skills are crucial to being successful.  When in doubt, over-communicate.  Write effective emails with useful subject lines and keep them brief.  A good email gets the message across quickly.  Be as transparent as possible in your communication – it will avoid the need to address the message again in the future.  Set and manage expectations and find the right communication tools for your team.  Be aware of time zones, you might be working with someone on the other side of the world.  Communicate time zones clearly and book meetings at reasonable times.

**Have a backup plan**

Working remotely is putting a tremendous amount of trust in technology.  But sometimes technology fails us.  You might wake up one day to no internet connection.  Have a backup plan; locate a coffee shop or library or have the ability to tether from your phone when the internet goes down.  The more prepared you are for the unknowns the better.

**Try new Things**

If something isn’t working, change it, try something new.  Be creative with how you approach working remotely.  Maybe the home office is too dull… spice it up with a new plant or other decorations.  Try working from a cafe from time to time – this can often provide a fresh perspective on the work day and be a good change of pace.

**Use the Right Tools for remote working**

Find the right tools that work for you and your team.  There’s an abundance of tools out there to improve communication, collaboration, productivity, and organization.  Here are some that might help:

[**Trello**](https://trello.com/) – Trello is a simple tool that can be used for everything from simple to-do lists to entire managing projects.

[**Wunderlist**](https://www.wunderlist.com/) – Wunderlist makes it easy to create shared to-do lists.

[**UberConference**](https://www.uberconference.com/) – UberConference is a light-weight conference call and screen-sharing solution.

[**Slack**](https://slack.com/) – Slack is the latest and greatest in real-time team communication.  It’s plug-and-play integrations allow you to connect many other tools to it.  Slack takes “virtual office” to a whole new level.

[**Noisli**](http://www.noisli.com/) – Noisli helps boost productivity by allowing you to mix different sounds to create your perfect work environment.

[**Workfrom**](https://workfrom.co/) – Workfrom is a guide to the best cafes, coffee shops, and bars for getting work done around the world.

We all work a little differently and every team encounters different challenges and solutions to their problems.  Talk to other remote workers for advice, share your own advice, and keep trying new things.  If you’re passionate enough about your job and focus on strong communication, your remote working experience should go pretty seamlessly.

For more information, tips, and best practices on working remotely, check out [Remote: Office Not Required](http://37signals.com/remote/), a book by the founders of Basecamp.

Advance your Career in Software Testing

After some time in an entry-level position in any career, a person will begin thinking about what comes next. Do I enjoy my work enough to continue down this path? Do I want to pursue something completely different? If you decide you enjoy your job and you want to continue with your current work, you may start to look at how to take it to the next level. What kind of promotions could you work towards? If the next level requires new responsibility or training, are you willing to put forth the effort? (Find out [How to begin your career in QA](https://blog.testlodge.com/how-to-begin-your-career-in-qa-testing/), if you are interested in software testing but have not yet started working in the sector.)

In the software testing profession, the typical progression is to move from an entry-level QA Engineer to a Senior QA Engineer (some companies use the title QA Analyst or other less used but equivalent titles). In some companies, there may be more than just junior and senior. For example, there might be QA Engineer I, II and III. If you want to progress in your software testing career, here is what you will need to do.

**Not All Titles are Equal**

Each organization has different criteria for job titles and the responsibilities tied to those titles. It’s important not to rely solely on your current employer’s definition of “Senior”, for example. Some employers may reserve the “Senior QA” title for automated testers. Others may base it largely on time served. It could also be a combination of things including the number of projects tested, time served and demonstrating leadership within projects. The important point is to remember that when you eventually move on to a new employer, you may find the skills that put you into your current senior role are not sufficient to land your next one. That could technically mean a demotion when changing jobs if you don’t keep yourself informed of the market expectations. Nobody wants to end up feeling pigeonholed in a company when it is time to move on.

**Choose a Direction**

When planning for career growth, it’s a good idea to have a direction in mind. With software testing, there are multiple avenues to choose from. Your desire might be to focus on test case development and manual testing, or you might have an interest in automated testing. Perhaps you would like to lead a QA team and ultimately become a QA manager? Maybe Web API testing interests you. By choosing a direction, you can then take steps to move towards your chosen goal.

If you decide you are interested in developing as an automated tester, you can look at the most popular automated testing tools and begin tutorials or [online courses](https://blog.testlodge.com/top-online-software-testing-courses/) that teach the skills necessary for using those tools. If management is something you would like to pursue, you might consider management related certifications and increase your knowledge of software development methodologies. Or perhaps it is too soon for you to make a determination. In that case you should take a wide swath and learn as many disciplines as possible. This is a good approach for anyone early in their career.

**Expand your QA Skill Set**

A key attribute of most Senior QA Engineers is their comprehensive knowledge of QA, which is needed for them to perform well in any specific software testing discipline. If you are currently focused on automation, it’s important to know [how to write test cases](https://blog.testlodge.com/how-to-write-test-cases-for-software-with-sample/) well. If you don’t understand tests case development and why they are written they way they are, it can be difficult to automate them. If you are only testing front-end applications, learn to test back-end services as well. An understanding of the system as a whole provides a tester with the insight necessary to do a better job within a specialty. Expanding your knowledge of aspects outside your specialty not only makes you more valuable within your current organization, but expanding your skillset will provide you with qualifications to get your next job when or if that time comes. The key here is to remain curious and proactive about learning disciplines outside of your specific job duties. Ask questions and learn from your teammates who are focused on other areas of testing.

**Look at the QA Market**

Once you have decided on a direction (and even if you haven’t), look at job postings for other QA positions. What are the qualifications required for those positions? You may find you haven’t heard of the tools or skills they are looking for in a candidate. Even if you aren’t looking to leave your job, there will be a wealth of information in those job postings, and you might see a pattern. For example, you might find Web API testing is a highly desired skill that is listed in almost all Senior QA openings. If it is not something that you have needed to test in your current position, it may be something to look into. Perhaps you can find out if there is a need within your current organization. This is a great way to figure out which skills you should focus on. You may find that the work you are currently doing is not sought after by other companies and it could pose a problem if you ever find yourself looking for a new job.

**Software Development Knowledge**

Another key attribute of a Senior Tester is a solid understanding of the software development process and the architecture behind the system under test. To lead testing of a project, for example, the senior tester should know the ins and outs of the entire process and the teams involved in making those changes. This includes understanding the effects of all development activities and how testing should be executed based on those activities. When certain back-end changes are made, a Senior Tester will understand what those changes mean for testing and where potential issues may arise.

**Conclusion**

The overarching theme here is to keep learning. Learn from your QA teammates, your development team and from sources like QA job postings. The worst thing that can happen is to stay inside a bubble and only focus on your day to day activities. Take time each day to invest in yourself and progress your career. It may seem like just a bit of additional work, but it is very rewarding. The knowledge gained will provide you with a sense of awareness of the industry you plan to excel in.

6 Ways to Drive Your QA Career Forward

Congratulations, you’ve just landed your first role as a Software Tester. What’s next, I hear you ask?



The job has started and you are gradually getting your feet under the desk. More than likely, you’ll be working in an agile environment because a majority of companies will work to some form of agile. As you’re sat at your desk, you might be thinking – I wonder what kind of things I can do to drive my career forward?

**Progressing Your QA Career**

**Commitment & Mindset**

Commitment to being a Software Tester is key to your future prospects and how you want to develop yourself. Even at this early stage, becoming committed to having a career in testing is the first challenge you will face. You now have opportunities to learn the foundations of testing that will take you to future roles, so having the mindset to realize the importance of “small things” and being someone who thinks outside the box will take you far. Train yourself to think like a user and remember you’re there to test the software’s integrity in advance of it becoming a live release.

**Knowledge**

Generally speaking, there will always be someone more knowledgeable about a specific area than yourself. You might have a Performance Tester on your team or an Automation Tester who writes in a language you’re keen to improve. Try to draw on their knowledge. Most likely, all you’ll have to do is ask about how they run tests or to show you how something works. You are on the same team and working to the same goals so they will be happy to help you. And don’t be afraid to ask the same questions again if you need to. You’re learning and picking up new technologies and they know that. There are extra ways to gain knowledge, of course. There are some good online learning resources such as Udemy available, as well as plenty of books out there for people who prefer reading.

**Qualifications**

You may hear colleagues or managers mentioning an ISEB or an ISTQB. These are certified testing qualifications that are often accessed as three-day courses followed by a 40-question multiple choice exam, which is then marked either as pass or fail. Having one of these qualifications on your CV can enhance your job prospects or even advance your career with your current employer but it won’t instantly turn you into a good tester. It’s what you do in the workplace that ultimately defines your abilities.

**Experience**

Gaining a broad experience in testing will help you to develop different skills and give you a better platform for building your career. Manual testing will help you to learn the front end of the software inside and out, and it can be an advantage if you have a good eye for detail. Most testing roles today require an element of automating because it saves time, so having automation experience on your CV can open up so many doors. Experience in other areas, such as performance testing or security testing can also be of benefit.

**Specialisation**

Specialising in a particular area allows you to increase knowledge and experience, so advancing your expertise. Although it’s helpful to know a bit about different areas, being a real expert in one or two could be more useful to those you work for and ultimately, for yourself.

**Team Management**

One day, you may have the opportunity to manage. If this happens, grab it with both hands. Even though it will be nerve wracking at first, if you have the talent and the people skills then gaining the respect of your team should come naturally. There are two ways to get into management. You could be hired into a team that has already been integrated into a business and progress from there, or you could be part of a growing business that eventually finds there is too much work for one person alone. Before you know it, what started with you as the sole tester is now a five-strong team with you in charge. At that point, your focus will be more on process than the actual software testing. You will also be concentrating on new ways to improve your team output so the group effort will still produce work at the same speed as when it was just you.

**Finally**

Being a tester isn’t easy. You have to take the rough with the smooth but primarily, you have to enjoy being a tester. A big part of the role is being passionate about the software you will be testing. It’s the best feeling in the world when a product goes live that you’ve slaved over and that without you, that moment would not have arrived. Having a career as a QA tester is not an easy path but with dedication, it can be very rewarding

Career Growth and Motivation for Software Testing Professionals

In any profession, a big key to enjoying your job is finding a way to stay motivated. Being challenged to learn new skills or to take on new responsibilities is important to both job satisfaction and career progression. In the software industry, developers, software testers and others involved in the development process must continuously stay up to date with the latest tools, languages, and processes so they can continue to excel in their jobs. Often times, moving with the industry is required just to stay employed.

This article is written specifically for those in the software testing world because as any tester knows, executing test cases can become tedious – especially when they are all done manually. Testers must be proactive when it comes to keeping themselves motivated so I’m going to talk about how to do exactly that.

**Tips for Staying Motivated**

**On the Job Training**

In some workplaces, the employer will periodically provide opportunities for growth, and you will regularly be challenged in ways that increase your skill set. This should be a priority for any company that values their employees and considers the long-term benefits of engaging their staff with positive challenges. A Quality Assurance manager should recognize the potential for testers to become burnt out from manual testing. If they don’t, they will burn through employees and the remaining testers will become less effective.

**Ask More of Your Employer**

Access at work to relevant training and challenges that promote growth should not be regarded as a benefit but a requirement.

In many workplaces, a new employee will learn new skills or tools to do their job, and then after the first six months to a year of employment, the growth slows significantly or stops. To continue their growth, the employee would need to change positions within the company, get a promotion or leave the company. There are many factors to consider when it comes to being happy in a job, but I believe being challenged on a consistent basis is absolutely necessary to feel the time spent on the job is not just a paycheck.

**Why does Professional Growth Slow?**

Part of the problem is the stream of constant project deadlines that don’t allow for research into improving process and efficiency. A good example of this occurs in software testing, specifically when the urgency to complete a manual testing software project outweighs the need to implement automation. When this happens once or twice to meet big deadlines, it’s understandable. Sometimes, tasks must be completed to satisfy a client or deliver on an important milestone. The real issue occurs when this happens repeatedly for many months or years leading employees to become burned out. It can be disheartening when the process that badly needs improvement will not become a priority any time soon. When testers suddenly realize they have been manually testing something that could have been automated far too long ago, their desire and interest fades away. They find it hard to continue manually testing the application with the same enthusiasm they had at the start of the job. If you feel like you have just been pushing buttons for months or years at this point, it is time to either push harder to improve the testing process or move on.

Automation is just one example of an improvement a software organization might repeatedly put off in favour of meeting regularly urgent deadlines and as a result, slow the growth of the employees. Other examples might include the time it takes to switch to a new tool, upgrade an existing platform or simply spend the money and time required to train for such a change.

**Conduct Research or Propose an Improvement**

So, before you up and leave the company in search of something that will scratch the itch to learn a new tool, language or process, perhaps you can do a bit of research and propose a change yourself. If the response is not positive or it becomes clear that there will likely not be any changes made to improve, then it’s probably time to look at options seriously.

**What Next?**

If you don’t feel like you are learning on the job or being given opportunities to expand your skill set, what should you do? Is it time to change positions for this reason alone? It might be – and I’ll explain why. There are ways to satisfy the need for professional growth that could allow you to stay where you are, but an employer must be open to you spending time on those endeavors, and it’s not always straightforward. The next step will be one or both of the following options:

**Growth from Change**

I have moved around fairly often in my software testing career. Over the past 12 years, I have worked at six organizations as a software tester, and my tenure within each company has ranged from 10 months to almost five years. At each place of employment, I have gained some extremely valuable experiences. While testing software has always been my primary job duty, each workplace provided a slightly different role, which required learning new skills and processes. I also found that each team I worked with offered a unique perspective on software development. There were similarities between each organization and certain differences that have helped me gain new perspectives. This factor alone has provided invaluable experience.

**Invest in Yourself**

The main thing to remember is that you need to invest in yourself continually. YOU are your best resource. If you are in a software testing job that isn’t providing the development opportunities you desire, make sure to keep up to date with what the market needs are. Periodically look at job postings, even if you have no plan to leave your job. You will be able to identify skills that are in demand and spend some time on these outside of your 9-5 job. Eventually, you may be able to use these skills in your current role or impress a prospective employer with your self-trained knowledge.

**Conclusion**

Set aside time each day for a bit of training for yourself. It doesn’t need to be a lot. Even 15 minutes a day working on a new skill could be extremely valuable. The key is consistency. Make it a priority to invest in yourself. While the work you do to invest in yourself may not deliver a paycheck today, it will provide a better paycheck in the future.